Ch. 9. Syntaxe SQL

De Apache OpenOffice Wiki
< FR | Documentation | HSQLDB Guide

Chapitre 9. Syntaxe SQL

Sommaire

Conventions de notations utilisées dans ce chapitre

[A] signifie que A est optionnel.

{ B | C } signifie que soit B, soit C doit être utilisé.

[{ B | C }] signifie que soit B, soit C peut-être utilisé, soit aucun des deux.

(et) sont les caractères '(' et ')' utilisés dans les instructions.

Les mots en MAJUSCULE sont des mot-clés

Commandes SQL

ALTER INDEX

```
<nomIndex> RENAME TO <nouveauNom>
```

Les noms d'index peuvent être modifiés tant qu'ils ne rentrent pas en conflit avec d'autres noms définis au niveau utilisateur ou système. [2]

ALTER SEQUENCE

```
<nomSéquence> RESTART WITH <valeur>
```

Exemple :

ALTER TABLE "TClients" ALTER COLUMN "RefClient" RESTART WITH 1

Ré-initialise la prochaine valeur retournée par la séquence. [2]

ALTER SCHEMA

```
<nomSchéma> RENAME TO <nouveauNom>
```

Attribue un nouveau nom au schéma. Tous les objets du schéma ne seront plus alors accessibles que par le nouveau nom du schéma. [2]

Nécessite les privilèges d'Administrateur.

ALTER TABLE

```
<nomTable> ADD [COLUMN] <nomColonne> Types de données
[(tailleColonne[,précision])] [{DEFAULT <valeurParDéfaut> |
GENERATED BY DEFAULT AS IDENTITY (START WITH <n>[, INCREMENT BY <m>])}] |
[[NOT] NULL] [IDENTITY] [PRIMARY KEY]
[BEFORE <colonneExistante>]
```

Ajoute une colonne à la fin de la liste des colonnes. L'option BEFORE <colonneExistante> permet de spécifier le nom d'une colonne existante afin que la nouvelle colonne soit insérée juste devant la <colonneExistante>. [2]

Cette instruction supporte une définition de colonne comme dans une commande CREATE TABLE. Si l'option NOT NULL est utilisée et que la table

n'est pas vide, une valeur par défaut doit être spécifiée. Cette condition mise à part, cette commande est l'équivalent d'une instruction de définition de colonne dans une instruction CREATE TABLE.

Si une vue SQL contient SELECT * FROM <nomTable> dans son instruction de sélection, la nouvelle colonne est ajoutée à la vue. Ceci est une fonctionnalité non standard qui sera probablement modifiée à l'avenir.

```
ALTER TABLE <nomTable> DROP [COLUMN] <nomColonne>
```

Supprime la colonne de la table. Toute contrainte de clé primaire ou d'unicité portant sur cette seule colonne sera également supprimée. Cette commande ne fonctionnera pas si la colonne fait partie d'une contrainte portant sur plusieurs colonnes ou si la colonne est référencée dans une contrainte de validation ou de clé étrangère.

Cette commande échouera également si la colonne est présente dans une vue SQL.

```
ALTER TABLE <nomTable> ALTER COLUMN <nomColonne> RENAME TO <nouveauNom>
```

Change le nom d'une colonne.

```
ALTER TABLE <nomTable> ALTER COLUMN <nomColonne> SET DEFAULT <valeurParDéfaut>
```

Attribue la valeur par défaut indiquée à la colonne. Utiliser NULL pour supprimer une valeur par défaut.

```
ALTER TABLE <nomTable> ALTER COLUMN <nomColonne> SET [NOT] NULL
```

Crée ou supprime une contrainte NOT NULL pour la colonne.

```
ALTER TABLE <nomTable> ALTER COLUMN <définitionDeColonne>
```

Cette forme de ALTER TABLE ALTER COLUMN accepte une définition de colonne comme dans une commande CREATE TABLE avec les restrictions suivantes.

Restrictions

- La colonne doit déjà être une clé primaire pour accepter une définition IDENTITY.
- Si la colonne est déjà une colonne IDENTITY et qu'il n'y a pas de définition pour IDENTITY dans la nouvelle définition de colonne, l'attribut IDENTITY existant est supprimé.

- L'expression de la valeur par défaut sera celle de la nouvelle définition de colonne, ce qui signifie qu'une valeur par défaut existante peut être supprimée en n'indiquant pas de nouvelle valeur par défaut, ou qu'une nouvelle valeur par défaut peut être spécifiée.
- L'attribut NOT NULL sera établi par la nouvelle définition de colonne (fonctionnement similaire au point précédent). En fonction des modifications exprimées, il est possible que la table doive être vide pour que la commande fonctionne.
- La commande fonctionne toujours si les changements exprimés sont possibles en général et que les valeurs déjà existantes dans la colonne peuvent toutes être converties. Par exemple, pour refaire une liste d'ID commençant à 1, sans "trous", la commande ci-dessous sera utilisée en remplaçant <nouvelle valeur séquence> par 1 :

```
ALTER TABLE <nomTable> ALTER COLUMN <nomColonne> RESTART WITH <nouvelle valeur séquence>
```

Cette forme est utilisée exclusivement pour les colonnes ayant l'attribut IDENTITY et modifie la prochaine valeur générée par la séquence identity.

```
ALTER TABLE <nomTable> ADD [CONSTRAINT <nomContrainte>] CHECK (<condition recherche>)
```

Ajoute une contrainte de validation à la table. Dans la version actuelle, une contrainte de validation ne peut faire référence qu'à la ligne insérée ou mise à jour.

```
ALTER TABLE <nomTable> ADD [CONSTRAINT <nomContrainte>] UNIQUE (<liste colonnes>)
```

Ajoute une contrainte d'unicité à la table. Cette commande ne fonctionne pas si il existe déjà une contrainte d'unicité comportant exactement la même <liste colonnes>. Cette commande ne fonctionne que si les valeurs de la liste des colonnes pour les lignes existantes sont uniques ou comportent une valeur null.

```
ALTER TABLE <nomTable> ADD [CONSTRAINT <nomContrainte>] PRIMARY KEY (<liste colonnes>)
```

Ajoute une contrainte de clé primaire à la table, avec la même syntaxe de contrainte que lors de la spécification d'une clé primaire dans une définition de table.

```
ALTER TABLE <nomTable>
ADD [CONSTRAINT <nomContrainte>] FOREIGN KEY (<liste colonnes>)
REFERENCES <tableRéférence> (<liste colonnes>)
[ON {DELETE | UPDATE} {CASCADE | SET DEFAULT | SET NULL}]
```

Ajoute une contrainte de clé étrangère à la table, avec la même syntaxe de contrainte que lors de la spécification d'une clé étrangère dans une définition

de table.

Cette commande échouera si il n'existe pas pour chaque ligne de la table <nomTable> une ligne correspondante (c'est à dire avec des valeurs identiques pour chaque colonne de la liste des colonnes) dans la table <tableRéférence>. Autrement dit, cette commande échouera s'il existe dans la table <nomTable> une ou plusieurs ligne(s) qui ne respecte(nt) pas la contrainte.

```
ALTER TABLE <nomTable> DROP CONSTRAINT <nomContrainte>
```

Supprime de la table une contrainte d'unicité, de validation ou de clé étrangère à laquelle le nom <nomContrainte> était attribué.

```
ALTER TABLE <nomTable> RENAME TO <nouveauNom>
```

ALTER USER

```
ALTER USER <nomUtilisateur> SET PASSWORD <motDePasse>
```

Change le mot de passe d'un uilisateur existant. Le mot de passe doit être entouré de guillemets doubles. Utiliser "" pour un mot de passe vide. [2]

Les utilisateurs disposant des droits d'administration peuvent modifier le nom de schéma par défaut pour un utilisateur avec la commande

```
ALTER USER <nomUtilisateur> SET INITIAL SCHEMA <nomSchema>
```

C'est le schéma par rapport auquel les noms des objets seront résolus pour cet utilisateur, sauf si cette règle est redéfinie ainsi que c'est expliqué dans . Pour des raisons de compatibilité avec les versions antérieures de HSQLDB, la valeur initiale du schéma ne sera pas conservée après une fermeture de la base avce les versions de HSQLDB inférieure à 1.8.1. (En d'autres termes, les paramètres INITIAL SCHEMA ne seront pas conservés après la fermeture de la base avec des versions de HSQLDB antérieures à la version 1.8.1).

Les privilèges d'administrateur sont requis pour utiliser ces commandes.

CALL

```
CALL Expression
```

N'importe quelle expression peut être appellée comme une procédure stockée, y-compris, mais pas seulement, des fonctions ou des procédures stockées Java. Cette commande renvoie un ResultSet d'une colonne et une ligne (le résultat),

exactement comme une requête SELECT d'une colonne et une ligne.

Voir aussi: Les procédures stockées / les fonctions., Expression SQL.

CHECKPOINT

```
CHECKPOINT [DEFRAG];
```

Ferme les fichiers de la base de données, écrit un nouveau fichier script, supprime le fichier log et rouvre la base de données. Si DEFRAG est spécifié, cette commande réduit la taille du fichier .data à sa taille minimale.

Voir aussi: SHUTDOWN, SET LOGSIZE.

COMMIT

```
COMMIT [WORK];
```

Termine une transaction et rend les modifications permanentes.

Voir aussi: ROLLBACK, SET AUTOCOMMIT, SET LOGSIZE.

CONNECT

```
CONNECT USER <nomUtilisateur> PASSWORD <motDePasse>;
```

Établit une connexion avec la base de données sous un autre nom d'utilisateur. Utiliser "" pour un mot de passe vide.

Voir aussi: GRANT, REVOKE.

CREATE ALIAS

```
CREATE ALIAS <fonction> FOR <javaFonction>;
```

Crée un alias pour une fonction Java statique destinée à être utilisée en tant que Procédure stockée. La fonction doit être accessible par la Machine Virtuelle Java dans laquelle la base de données est exécutée. Exemple:

```
CREATE ALIAS ABS FOR "java.lang.Math.abs";
```

Note

La commande CREATE ALIAS ne fait que définir l'alias. Elle ne valide pas l'existence de la méthode cible ou de la classe qui la contient. Pour valider

l'alias, utilisez-le.

Voir aussi: CALL, Procédures stockées / Fonctions.

CREATE INDEX

```
CREATE [UNIQUE] INDEX <index> ON <table> (<colonne> [DESC] [, ...]) [DESC];
```

Crée un index sur une ou plusieurs colonnes d'une table. La création d'un index sur des colonnes fréquemment incluse dans des recherches peut améliorer les performances. Le qualificatif DESC peut être utilisé pour des raisons de compatibilité de la commande avec d'autres bases de données, mais il n'a aucun effet. Des index d'unicité peuvent être définis de cette manière, mais cela est périmé. Il vaut mieux utiliser des contraintes UNIQUE à la place. Le nom d'un index doit être unique pour toute la base de données.

Voir aussi: CREATE TABLE, DROP INDEX.

CREATE ROLE

```
CREATE ROLE <nomRole>;
```

Crée le rôle <nomRole> sans membres. Nécessite les privilèges d'administrateur. [2]

CREATE SCHEMA

```
CREATE SCHEMA <nomSchéma> AUTHORIZATION <bénéficiaire>
    [<expressionDeCréation> [<expressionDeBénéficiaire>] [...];
```

Crée le schéma <nomSchéma> dont le propriétaire est le bénéficiaire spécifié. Le bénéficiaire de l'autorisation peut être un utilisateur ou un rôle. [2]

Les instructions (imbriquées) optionnelles CREATE et GRANT ne peuvent être affectées qu'à de nouveaux objets dans ce nouveau schéma. Seule la dernière instruction imbriquée doit être terminée par un point-virgule car le premier point-virgule rencontré après "CREATE SCHEMA" termine la commande CREATE SCHEMA. Dans l'exemple ci-dessous, un nouveau schéma, ACCOUNTS, est créé, ensuite deux tables et une vue sont ajoutées à ce schéma et certains droits sur ces objets sont attribués.

```
CREATE SCHEMA ACCOUNTS AUTHORIZATION DBA
   CREATE TABLE AB(A INTEGER, ...)
   CREATE TABLE CD(C CHAHR, ...)
   CREATE VIEW VI AS SELECT ...
   GRANT SELECT TO PUBLIC ON AB
   GRANT SELECT TO JOE ON CD;
```

Notez que cet exemple consiste en une instruction CREATE SCHEMA qui se termine par un point-virgule.

Nécessite les privilèges d'administrateur.

CREATE SEQUENCE

```
CREATE SEQUENCE <nomSéquence> [AS {INTEGER | BIGINT}]
   [START WITH <valeurDébut>] [INCREMENT BY <valeurIncrémentation>];
```

Crée une séquence. Le type par défaut est INTEGER. La valeur de début par défaut est 0 et l'incrémentation par défaut est 1. Les valeurs négatives ne sont pas acceptées. Si une séquence dépasse Integer.MAXVALUE ou Long.MAXVALUE, le résultat suivant est déterminé par l'arithmétique en complément à deux. [2]

La prochaine valeur d'une séquence peut être incluse dans une instruction SELECT, INSERT ou UPDATE comme dans l'exemple suivant:

```
SELECT [...,] NEXT VALUE FOR <nomSéquence> [, ...] FROM <nomTable>;
```

Dans la version SQL 200n à l'étude, et dans la version actuelle, il n'est pas possible de retrouver la dernière valeur retournée par une séquence.

CREATE TABLE

Exemples de code :

```
CREATE TABLE  T_Classifications  ("ID_classification" INTEGER  IDENTITY , "Classification" VARCHAR(20
```

```
CREATE TABLE  T_Employe ("ID_Employe" INTEGER IDENTITY , "Nom" VARCHAR(30), "Prenom" VARCHAR(30), "Da
"Date_debut_contrat" DATE, "ID_Ville" VARCHAR(3))
```

Structure générale :

```
CREATE [MEMORY | CACHED | [GLOBAL] TEMPORARY | TEMP | TEXT] TABLE <nom>
   ( <columnDefinition> [, ...] [, <définitionDeContrainte>...] )
   [ON COMMIT {DELETE | PRESERVE} ROWS];
```

Crée une table en mémoire (par défaut) ou sur le disque avec seulement un cache en mémoire. Si la base de données est spécifiée comme en mémoire seulement, les deux formes MEMORY et CACHED de CREATE TABLE créent une table en mémoire tandis que la forme TEXT n'est pas autorisée.

Composants d'une commande CREATE TABLE

Définition de colonne

```
définitionDeColonne nomColonne typeDonnées [(tailleColonne[,précision])]
   [{DEFAULT <valeurParDéfaut> | GENERATED BY DEFAULT AS IDENTITY
   (START WITH <n>[, INCREMENT BY <m>])}] |
   [[NOT] NULL] [IDENTITY] [PRIMARY KEY]
```

Les valeurs par défaut autorisées sont les constantes ou certaines fonctions SQL dates et heures.

Valeurs par défaut permises dans les définitions de colonnes:

- Pour les colonnes de type caractères, une chaîne de caractères entourée de guillemets simples ou NULL. La seule fonction SQL autorisée est CURRENT_USER.
- Pour les colonnes de type dates et heures, une valeur de DATE, TIME ou TIMESTAMP entourée de guillemets simples, ou NULL. Ou bien une fonction SQL dates et heures telle que CURRENT_DATE, CURRENT_TIME, CURRENT_TIME, CURRENT_TIMESTAMP, TODAY, NOW. La fonction utilisée doit retourner le type de données de la colonne.
- Pour les colonnes de type BOOLEAN, les littéraux FALSE, TRUE, NULL.
- Pour les colonnes de type numérique, un nombre valide ou NULL.
- Pour les colonnes de type binaire, une chaîne hexadécimale valide ou NULL. Une seule colonne avec l'attribut IDENTITY est autorisée par table.

Les colonnes avec l'attribut IDENTITY sont auto-incrémentées. Elles doivent être de type INTEGER ou BIGINT et sont automatiquement définies comme clé primaire (en conséquence, les clés primaires sur plusieurs colonnes ne sont pas autorisées lorsque une colonne IDENTITY est présente dans la table). Lorsque l'on utilise la syntaxe SQL longue, la clause (START WITH <n>) spécifie la première valeur qui sera utilisée. La dernière valeur insérée dans une colonne IDENTITY durant la connexion en cours peut être obtenue avec la fonction IDENTITY(). Par exemple, si Id est une colonne IDENTITY:

```
INSERT INTO Test (Id, Name) VALUES (NULL,'Test');
   CALL IDENTITY();
```

Définition de contrainte

```
[CONSTRAINT <nom>]
   UNIQUE ( <colonne> [,<colonne>...] ) |
   PRIMARY KEY ( <colonne> [,<colonne>...] ) |
```

```
  FOREIGN KEY ( <colonne> [,<colonne>...] )
  REFERENCES <tableRéférencée> ( <colonne> [,<colonne>...])
  [ON {DELETE | UPDATE}
  {CASCADE | SET DEFAULT | SET NULL}] |
  CHECK(<condition validation>)
```

Les deux clauses ON DELETE et ON UPDATE peuvent être utilisées dans la même définition de clé étrangère.

Condition de recherche

Une condition de recherche est similaire à un ensemble de conditions dans une clause WHERE. Dans la version actuelle de HSQLDB, les conditions d'une contrainte CHECK ne peuvent faire référence qu'à la ligne en cours de traitement, ce qui veut dire qu'elles ne peuvent contenir une instruction SELECT. Des exemples de définitions de tables comportant des contraintes CHECK sont disponibles dans TestSelfCheckConstraints.txt. Ce fichier est dans le répertoire /hsqldb/testrun/hsqldb/ de la distribution.

Limitations générales de la syntaxe

Les bases de données HSQLDB sont créées dans un mode compatible avec les versions antérieures qui n'impose pas le respect de la taille et de la précision des colonnes. Vous pouvez assigner une valeur true à la propriété: sql.enforce_strict_size=true pour activer cette fonctionalité. Lorsque cette propriété est établie, toute indication de taille et de précision pour une colonne de type numérique ou caractères est appliquée. Utilisez la commande SET PROPERTY "sql.enforce_strict_size" TRUE une seule fois avant de définir les tables.

Les contraintes NOT NULL doivent faire partie de la définition de colonne. Les autres contraintes ne peuvent faire partie de la définition de colonne et doivent être placées après la liste des définitions de colonnes.

Le contenu des tables temporaires TEMPORARY TABLE est supprimé par défaut à chaque COMMIT ou ROLLBACK. L'option ON COMMIT PRESERVE ROWS permet de garder le contenu des lignes pour la durée de la connexion. La valeur par défaut est ON COMMIT DELETE ROWS

Voir aussi: DROP TABLE.

CREATE TRIGGER

```
CREATE TRIGGER <nom> {BEFORE | AFTER} {INSERT | UPDATE | DELETE} ON <table>
    [FOR EACH ROW] [QUEUE n] [NOWAIT] CALL <TriggerClass>;
```

TriggerClass est une classe fournie par l'application qui implémente l'interface org.hsqldb.Trigger , par exemple MonPackage.MaClasseTrigger. C'est la méthode fire() de cette classe qui est appelée lorsqu'un évènement déclencheur est généré. C'est à vous de fournir cette classe, qui peut porter un

nom au choix, et vous devez vous assurer que cette classe est présente dans le classpath utilisé au démarrage d'hsqldb.

Depuis la version 1.7.2, l'implémentation a été modifiée et améliorée. Lorsque la méthode 'fire' est appellée, les arguments suivants lui sont passés:

```
fire (String nom, String table, Object ligne1[], Object ligne2[])
```

où 'ligne1' et 'ligne2' représentent l'état de la ligne sur laquelle on agit 'avant' et 'après' l'instruction, chaque colonne étant un élément du tableau. La correspondance entre les éléments du tableau et les types de données de la base est spécifiée dans . Par exemple, BIGINT est représenté par un Objet java.lang.Long. Notez bien que le nombre d'éléments dans les tableaux qui représentent une ligne peut être supérieur d'un ou deux éléments au nombre de colonnes. Ne modifiez jamais les derniers éléments des tableaux, qui ne font pas réellement partie de la véritable ligne .

Si la méthode de déclenchement veut accéder à la base de données, elle doit établir sa propre connexion JDBC. Cela peut provoquer des incohérences dans les données et d'autres problèmes, ce n'est donc pas recommandé. L'URL jdbc:default:connection: n'est pas supportée actuellement.

Note sur l'implémentation :

Si QUEUE 0 est spécifié, la méthode fire() est exécutée dans le même thread que le moteur de base de données. Cela permet à l'action déclenchée de modifier les données qui vont être enregistrées dans la base. Cela permet à l'action déclenchée par le déclencheur de modifier les données qui vont être insérées dans la base de données. Les données peuvent être vérifiées ou modifiées lors d'un déclencheur BEFORE INSERT / UPDATE + FOR EACH ROW. Toutes les contraintes de la table sont alors appliquées par le moteur de base de données, et toute violation sera rejetée pour la commande SQL qui a initié un INSERT ou UPDATE. Il-y-a une exception à cette règle : lors d'une requête UPDATE, la vérification de l'intégrité référentielle et les actions en cascade liées à ON UPDATE CASCADE / SET NULL / SET DEFAULT sont exécutées avant l'appel de la méthode du déclencheur. Si une valeur non valide qui ne respecte pas l'intégrité référentielle est insérée dans la ligne par la méthode du déclencheur, cette insertion n'est pas vérifiée et peut conduire à l'enregistrement de données incohérentes dans la table.

Par contre, si le déclencheur est utilisé à des fins de communication externe, et non pas pour vérifier ou modifier les données, une taille de file d'attente supérieure à 0 peut être spécifiée. Cela a l'avantage de ne pas bloquer le thread principal de

la base de données car chaque déclencheur est exécuté dans son propre thread qui attend que l'évènement déclencheur se produise. Lorsque cela se produit, le thread du déclencheur appelle la méthode TriggerClass.fire(). Il-y-a alors une file d'attente d'évènements qui attendent d'être interceptés par les différents thread de déclencheurs. Cela est particulièrement utile dans le cas de déclencheurs 'FOR EACH ROW', lorsqu'un grand nombre d'évènements générés par des déclencheurs se succèdent rapidement, sans que le thread du déclencheur aie l'occasion de s'exécuter. Si la file d'attente est pleine, l'ajout d'un nouvel évènement dans la file provoque une pause du moteur de base de données dans l'attente d'espace libre dans la file. Prenez soin d'éviter ce cas de figure lorsque les méthodes déclenchées par les déclencheurs doivent accéder à la base de données car le programme sera bloqué. Cela peut être évité soit en attribuant au paramètre QUEUE une valeur suffisamment grande, soit en utilisant le paramètre NOWAIT, qui impose le replacement de l'évènement déclencheur le plus récent par le nouvel évènement dans la file d'attente. La valeur par défaut de la taille de la file d'attente est 1024. Notez également que la chronologie de l'appel des méthodes de déclencheur n'est pas garantie, les applications doivent donc implémenter leurs propres moyens de synchronisation si nécessaire.

Lorsque le paramètre QUEUE a une valeur différente de zéro, si la méthode déclenchée par le déclencheur modifie la valeur de 'ligne2', les modifications peuvent aussi bien être enregistrées dans la base que non prises en compte, et cela produira presque surement des incohérences dans les données.

Consultez le code des classes org.hsqldb.sample.Trigger (http://hsqldb.org/doc/src/org/hsqldb/Trigger.html%7C) and org.hsqldb.sample.TriggerSample (http://hsqldb.org/doc/src/org/hsqldb/sample/TriggerSample.html%7C) pour plus d'informations sur la manière d'écrire des classes déclencheur.

Voir aussi: DROP TRIGGER.

CREATE USER

```
CREATE USER <nomUtilisateur> PASSWORD <motDePasse> [ADMIN];
```

Crée un nouvel utilisateur ou un nouvel administrateur pour cette base de données. Le mot de passe doit être entouré de guillemets doubles. Vous pouvez créer un mot de passe vide avec "". Il est possible de modifier un mot de passe ultérieurement en utilisant la commande ALTER USER.

Nécessite les privilèges d'administrateur.

Voir aussi :CONNECT, GRANT, REVOKE. ALTER USER[2],

CREATE VIEW

```
CREATE VIEW <nomVue>[(<aliasColonneVue>,..) AS SELECT ... FROM ... [WHERE Expression]
   [ORDER BY expressionDeTri [, ...]]
   [LIMIT <limite> [OFFSET <offset>]];
```

Une vue peut être considérée soit comme une table virtuelle, soit comme une requête enregistrée. Les données accessibles à travers une vue ne sont pas enregistrées dans la base comme objets distincts. Ce qui est enregistré dans la base de données est une instruction SELECT. Le résultat de l'instruction SELECT constitue la table virtuelle présentée par la vue. Un utilisateur peut exploiter cette table virtuelle en faisant référence à la vue par son nom dans des instructions SQL de la même manière qu'il fait référence à une table. Une vue peut être utilisée pour l'un ou l'autre de ces différents usages :

- Restreindre l'accès d'un utilisateur à certaines lignes d'une table. Par exemple, permettre à un employé d'accéder aux seules lignes le concernant dans une table de suivi de tâches.
- Restreindre l'accès d'un utilisateur à certaines colonnes d'une table. Par exemple, permettre à un employé qui ne travaille pas au service de la paie de consulter le nom, le service et le numéro de poste dans une table Employés mais lui interdire l'accès aux données personnelles ou aux informations de rémunération.
- Rassembler des colonnes issues de plusieurs tables afin que l'utilisateur ne voie qu'une table.
- Réaliser des fonctions d'agrégation statistique. Par exemple, présenter la somme des valeurs d'une colonne, ou bien la plus grande ou la plus petite valeur d'une colonne.

Les vues sont créées en définissant l'instruction SELECT qui détermine les données à afficher dans la vue. Les tables référencées par l'instruction SELECT sont appelées les tables sources de la vue. Dans cet exemple, mealsjv est une vue dont les données sont issues de trois tables sources pour présenter une table virtuelle de données fréquemment utilisées :

```
CREATE VIEW mealsjv AS
    SELECT m.mid mid, m.name name, t.mealtype mt, a.aid aid,
           a.gname + ' ' + a.sname author, m.description description,
           m.asof asof
      FROM meals m, mealtypes t, authors a
     WHERE m.mealtype = t.mealtype
       AND m.aid = a.aid;
```

Vous pouvez ensuite faire référence à mealsjv dans des instructions de la même manière que vous pouvez faire référence à une table quelconque :

```
SELECT * FROM mealsjv;
```

Une vue peut faire référence à une autre vue. Par exempel, mealsjv présente des informations qui peuvent être utiles pour des descriptions longues qui contiennent des identifiants, mais pour une page web, une présentation plus concise pourrait être suffisante. On peut créer une vue qui ne présente que certaines colonnes de mealsjv :

```
CREATE VIEW mealswebv AS SELECT name, author FROM mealsjv;
```

L'instruction SELECT dans une définition de vue doit retourner des noms de colonnes distincts. Si deux colonnes dans l'instruction SELECT ont le même nom, utilisez des alias de colonnes pour les distinguer l'une de l'autre. Il est toujours possible de définir de nouveaux noms de colonnes dans une vue.

```
CREATE VIEW aview (new_name, new_author) AS SELECT name, author FROM mealsjv
```

Voir aussi :Expression SQL, SELECT[2], DROP VIEW[2].

DELETE

```
DELETE FROM table [WHERE Expression];
```

Supprime des lignes dans une table.

Voir aussi : Expression SQL, INSERT, SELECT[2].

DISCONNECT

```
DISCONNECT;
```

Ferme la connexion en cours. Il n'est pas nécessaire d'appeler cette commande lorque vous utilisez l'interface JDBC : elle est appelée automatiquement lorsque la connexion est fermée. Après s'être déconnecté, il n'est plus possible d'exécuter d'autres requêtes (y-compris CONNECT) avec cette connexion.

Voir aussi : CONNECT.

DROP INDEX

```
DROP INDEX index [IF EXISTS];
```

Supprime l'index spécifié de la base de données. Ne fonctionnera pas si l'index est utilisé pour une contrainte UNIQUE ou FOREIGN KEY.

Voir aussi : CREATE INDEX.

DROP ROLE

```
DROP ROLE <nomRole>;
```

Supprime tous les membres du rôle spécifié, puis supprime le rôle lui-même.[2]

DROP SEQUENCE

```
DROP SEQUENCE <nomSequence> [IF EXISTS] [RESTRICT | CASCADE];
```

Supprime la séquence spécifiée de la base de données. Si l'option IF EXISTS est utilisée, l'instruction ne renvoie pas d'erreur si la séquence n'existe pas. L'option RESTRICT est active par défaut, elle signifie que la commande DROP échouera si une vue fait référence à la séquence. Utilisez l'option CASCADE pour supprimer tous les objets de la base de données qui dépendent de la séquence supprimée.[2]

DROP SCHEMA

```
DROP SCHEMA <nomSchema> [RESTRICT | CASCADE];
```

Supprime le schéma spécifié de la base de données. L'option RESTRICT est active par défaut, elle signifie que la commande DROP échouera si des objets tels que des tables ou des séquences ont été définis dans ce schéma. Utilisez l'option CASCADE pour supprimer tous les objets du schéma. [2]

Nécessite les privilèges d'administrateur.

DROP TABLE

```
DROP TABLE <table> [IF EXISTS] [RESTRICT | CASCADE];
```

Supprime une table, les données et les index de la base de données. Si l'option IF EXISTS est utilisée, l'instruction ne renvoie pas d'erreur même si la table n'existe pas. L'option RESTRICT, qui est active par défaut, signifie que la commande DROP échouera si des tables ou des vues font référence à cette table. Utilisez l'option CASCADE pour supprimer toutes les vues qui font référence à cette table, ainsi que toutes les contraintes de clé étrangère qui relient cette table à d'autres tables.

Voir aussi : CREATE TABLE.

DROP TRIGGER

```
DROP TRIGGER <trigger>;
```

Supprime un déclencheur de la base de données.

Voir aussi : CREATE TRIGGER[2].

DROP USER

```
DROP USER <nomUtilisateur>;
```

Supprime un utilisateur de la base de données.

Nécessite les privilèges d'administrateur.

Voir aussi : CREATE USER.

DROP VIEW

```
DROP VIEW <nomVue> [IF EXISTS] [RESTRICT | CASCADE];
```

Supprime une vue de la base de données. Si l'option IF EXISTS est utilisée, l'instruction ne renvoie pas d'erreur même si la vue n'existe pas. L'option RESTRICT est active par défaut et signifie que la commande DROP échouera si une autre vue fait référence à la vue supprimée. Utilisez l'option CASCADE pour supprimer toutes les vues dépendantes de cette vue. [2]

Voir aussi : CREATE VIEW[2].

EXPLAIN PLAN

```
EXPLAIN PLAN FOR { SELECT ... | DELETE ... | INSERT ... | UPDATE ..};
```

EXPLAIN PLAN FOR peut être utilisé avec n'importe quelle requête pour obtenir une liste détaillée des éléments utilisés pour le plan d'exécution.

Cette liste comprend les index utilisés pour exécuter la requête et peut être utilisée pour optimiser la requête ou rajouter des index à des tables.

GRANT

```
GRANT { SELECT | DELETE | INSERT | UPDATE | ALL } [,...]
ON { table | CLASS "package.class" } TO <beneficiaire>;
```

```
GRANT <nomRole> [,...] TO <beneficiaire>;
```

<beneficiaire> peut être un nom d'utilisateur, un nom de rôle, ou PUBLIC. PUBLIC signifie tous les utilisateurs.

La première forme de la commande GRANT attribue des privilèges à un bénéficiaire sur une table ou sur une classe. Pour autoriser un utilisateur à appeller une fonction statique d'une procédure stockée, le droit ALL doit être utilisé. Exemples :

```
GRANT SELECT ON Test TO GUEST;
GRANT ALL ON CLASS "java.lang.Math.abs" TO PUBLIC;
```

```
   Attention: Même si la commande est GRANT ALL ON CLASS, vous devez spécifier un nom de méthode stat
   Vous attribuez un droit d'accès à une méthode, pas à une classe.
```

La seconde forme de la commande GRANT inscrit le <beneficiaire> dans le rôle spécifié.

Nécessite les privilèges d'administrateur.

Voir aussi : REVOKE, CREATE USER, CREATE ROLE [2].

INSERT

```
INSERT INTO table [( nomColonne [,...] )]
{ VALUES(Expression [,...]) | ExpressionDeSelection};
```

Ajoute une ou plusieurs lignes de données dans une table.

Exemple de commande à saisir en SQL direct :

```
INSERT INTO "Clients" ("Nom", "Prénom")
SELECT "Nom", "Prénom"  FROM "AjoutClients"
WHERE NOT "AjoutClients"."ID" IN ( SELECT "Clients"."ID" FROM "Clients" )
```

REVOKE

```
REVOKE { SELECT | DELETE | INSERT | UPDATE | ALL } [,...]
ON { table | CLASS "package.class" } FROM <beneficiaire>;
```

```
REVOKE <nomRole> [,...] FROM <beneficiaire>;
```

<beneficiaire> peut être un nom d'utilisateur, un nom de rôle ou PUBLIC. PUBLIC signifie tous les utilisateurs.

La première forme de la commande REVOKE supprime les privilèges d'un bénéficiaire sur une table ou une classe.

La seconde forme de la commande REVOKE supprime l'inscription d'un <beneficiaire> du rôle spécifié.

Les deux formes nécessitent les privilèges d'administrateur.

Voir aussi : GRANT.

ROLLBACK

```
ROLLBACK [TO SAVEPOINT <nomPointDeSauvegarde> | WORK}];
```

ROLLBACK, utilisé seul ou en combinaison avec WORK, annule les changements effectués depuis le dernier COMMIT ou ROLLBACK.

ROLLBACK TO SAVEPOINT <nomPointDeSauvegarde> annulle les changements effectués depuis le point de sauvegarde spécifié. Cette commande n'a pas d'effet si le point de sauvegarde spécifié n'est pas trouvé.

Voir aussi : COMMIT.

SAVEPOINT

```
SAVEPOINT <nomPointDeSauvegarde>;
```

Crée un SAVEPOINT pour utiliser avec ROLLBACK TO SAVEPOINT.

Voir aussi : COMMIT.

SCRIPT

```
SCRIPT ['fichier'];
```

Crée un script SQL de description de la base de données. Si le fichier n'est pas spécifié, la commande retourne un result set contenant le DDL. Si le fichier est spécifié, alors le fichier est sauvegardé avec le chemin relatif à la machine où le moteur de base de données est exécuté.

Nécessite les privilèges d'administrateur.

SELECT

```
SELECT [{LIMIT <offset> <limite> | TOP <limite>}][ALL | DISTINCT]
{ expressionDeSelection | table.* | * } [, ...]
[INTO [CACHED | TEMP | TEXT] nouvelleTable]
FROM listeDeTables
[WHERE Expression]
[GROUP BY Expression [, ...]]
[HAVING Expression]
[{ UNION [ALL | DISTINCT] | {MINUS [DISTINCT] | EXCEPT [DISTINCT] } |
INTERSECT [DISTINCT] } selectStatement]
[ORDER BY expressionDeTri [, ...]]
[LIMIT <limite> [OFFSET <offset>]];
```

Recherche des données issues d'une ou plusieurs tables de la base de données. [2]

Composants d'une commande SELECT:

listeDeTables

```
  table [{CROSS | INNER | LEFT OUTER | RIGHT OUTER}
  JOIN table ON Expression] [, ...]
```

table

```
  { (selectStatement) [AS] etiquette | nomTable}
```

expressionDeSelection

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

- Rechercher l'enregistrement le plus récent avec MAX (http://user.services.openoffice.org/fr/forum/ftopic33316.html)

Si DISTINCT est utilisé, une seule instance de valeurs équivalentes est utilisée dans la fonction d'aggrégation. Mis à part COUNT(*), toutes les fonctions d'aggrégation ignorent les valeurs NULL. Le type des données retournées par SUM est sujet à un transtypage déterministe éventuel pour garantir des résultats sans erreurs. Le type des données retournées par COUNT est INTEGER, pour MIN, MAX et AVG le type des données retournées est le même que celui des colonnes concernées. Pour SOME et EVERY, le type retourné est BOOLEAN. Pour les fonctions statistiques VAR_POP, VAR_SAMP, STDDEV_POP et STDDEV_SAMP, le type retourné est toujours DOUBLE. Ces fonctions statistiques ne permettent pas l'utilisation des qualificateurs ALL ou DISTINCT.

Si CROSS JOIN est utilisé, il n'est pas permis d'utiliser une expression ON pour la jointure.

expressionDeTri

```
  { NoColonne | aliasColonne | expressionDeSelection }
        [ASC | DESC]
```

LIMIT n m

Crée d'abord le result set pour l'instruction SELECT puis supprime les n premières lignes (OFFSET) et retourne les m premières lignes du result set ainsi obtenu (LIMIT). cas particuliers : LIMIT 0 m est équivalent à TOP m ou FIRST m dans d'autres systèmes de gestions de bases de donées; LIMIT n 0 supprime les n premières lignes du result set et renvoie toutes les lignes restantes.

LIMIT m OFFSET n

Cette forme est utilisée à la fin de l'instruction SELECT. OFFSET est optionnel.

TOP m

Équivalent à LIMIT 0 m.

UNION et autres opérations sur les ensembles

Les instructions SELECT multiples jointes avec UNION, EXCEPT et INTERSECT sont possibles. Chaque SELECT est alors considéré comme un terme, et l'opération sur les ensembles comme un opérateur dans une expression. L'expression est évaluée de gauche à droite mais INTERSECT bénéficie d'une précédence sur les autres opérateurs et est appliqué en premier. Vous pouvez utiliser des parenthèses autour des instructions SELECT pour modifier l'ordre d'évaluation.

- Faire une Somme par colonne avec UNION (https://forum.openoffice.org/fr/forum/ftopic32757.html#p326081)

Voir ausi : INSERT, UPDATE, DELETE.

SET AUTOCOMMIT

```
SET AUTOCOMMIT { TRUE | FALSE };
```

Active ou désactive le mode auto-commit de la connexion. Si le mode auto-commit est activé, toutes les instructions seront exécutées en tant que transactions individuelles. Lorsque le mode auto-commit est désactivé, les instructions sont regroupées en transaction qui sont finalisées soit par

COMMIT , soit par ROLLBACK. Par défaut, les nouvelles connexions sont en mode auto-commit. Cette commande ne devrait pas être utilisée directement, utilisez plutôt la méthode JDBC équivalente : Connection.setAutoCommit(boolean autocommit).

SET DATABASE COLLATION

```
SET DATABASE COLLATION <nomCollationEntreGuillemetsDoubles;
```

Chaque base de données peut avoir sa propre collation. Attribue une collation parmi les collations proposées dans le code source de org.hsqldb.Collation.

Lorsque cette commande a été exécutée, la base peut être ouverte dans n'importe quel environnement d'exécution Java et elle conservera sa collation.

SET CHECKPOINT DEFRAG

```
SET CHECKPOINT DEFRAG <taille>;
```

Le paramètre taille représente le nombre de mega-octets perdus dans le fichier .data. Lorsque CHECKPOINT est exécuté, soit parce que le fichier .log a atteint la limite de taille fixée par "SET LOGSIZE taille", soit parce que l'utilisateur exécute une commande CHECKPOINT, l'espace perdu durant la session est mesuré et si il est supérieur à taille, un CHECKPOINT DEFRAG est exécuté au lieu de CHECKPOINT.

SET IGNORECASE

```
SET IGNORECASE { TRUE | FALSE };
```

Désactive (ignorecase = true) ou active (ignorecase = false) la sensibilité à la casse de la comparaison de texte et de l'indexation pour les nouvelles tables. Par défaut, les colonnes de caractères dans les nouvelles bases de données sont sensibles à la casse. La sensibilité à la casse doit être précisée avant la création des tables. Les tables existantes et leurs données ne sont pas affectées lors d'un changement de sensibilité à la casse. Lorsque vous choisissez d'ignorer la casse, les colonnes de type VARCHAR sont créées en tant que VARCHAR_IGNORECASE dans les nouvelles tables. Il est également possible de spécifier le type de données VARCHAR_IGNORECASE lors de la définition de colonne. Cela permet d'avoir des colonnes sensibles à la casse et d'autres non, y-compris dans la même table.

Nécessite les privilèges d'administrateur.

SET INITIAL SCHEMA

Les utilisateurs peuvent modifier le nom de schéma par défaut de leur base de données avec la commande

```
SET INITIAL SCHEMA <nomSchema>;
```

C'est le schéma par rapport auquel les noms d'objets de la base de données seront résolus pour l'utilisateur en cours, sauf si ce choix est outrepassé comme expliqué dans Nomenclature des objets de schéma. Pour des raisons de rétro-compatibilité, la valeur du schéma initial ne sera pas conservée au cours des fermetures de la base de données jusqu'à la version 1.8.1 de HSQLDB. (par ex. les réglages de INITIAL SCHEMA seront perdus sur fermeture de la base de données avec des versions de HSQLDB antérieures à la 1.8.1).

SET LOGSIZE

```
SET LOGSIZE <taille>;
```

Définit la taille maximum en Mo du fichier .log. La valeur par défaut est 200 Mo. La base de données sera fermée puis ré-ouverte (comme en utilisant CHECKPOINT) si la taille du fichier .log dépasse cette limite, la taille du fichier .log étant ainsi réduite. Zéro signifie pas de limite de taille.

Voir aussi: CHECKPOINT.

SET MAXROWS

```
SET MAXROWS <maxwors>;
```

Describe me!

SET PASSWORD

```
SET PASSWORD <motDePasse>;
```

Modifie le mot de passe de l'utilisateur connecté. Le mot de passe doit être entouré de guillemets doubles. Vous pouvez attribuer un mot de passe vide avec "".

SET PROPERTY

```
SET PROPERTY <nomEntreGuillemetsDoubles> <valeur>;
```

Attribue une valeur à une propriété de la base de données. Les propriétés qui peuvent être modifiées en utilisant cette commande sont indiquées dans le chapitre 4.

SET READONLY

```
SET READONLY {TRUE|FALSE};
```

Describe me!

SET REFERENTIAL_INTEGRITY

```
SET REFERENTIAL_INTEGRITY { TRUE | FALSE };
```

Cette commande active / désactive le contrôle de l'intégrité référentielle (clés étrangères). Normalement, ce contrôle doit être activé (c'est la valeur par défaut), mais lors de l'importation de données, si les données sont importées dans le désordre, le contrôle peut être désactivé.

Attention: Notez que lorsque le contrôle de l'intégrité référentielle est rétabli, aucun contrôle de l'intégrité référentielle des données modifiées entretemps n'est réalisé. Vous pouvez vérifier la cohérence des données en utilisant des requêtes SQL et effectuer les actions appropriées.

Nécessite les privilèges d'administrateur.

Voir aussi : CREATE TABLE.

SET SCHEMA

```
SET SCHEMA <nomSchema>;
```

Spécifie le schéma utilisé par la session JDBC en cours. Le seul objectif du schéma de session est de fournir un nom de schéma par défaut pour les objets de schéma dont le nom de schéma n'est pas explicitement spécifié dans la commande SQL ou par association avec un autre objet dont le schéma est connu. Par exemple, si vous exécutez l'instruction SELECT * FROM atbl;, HSQLDB recherchera la table ou la vue nommée atbl dans le schéma de la session en cours.

Les schémas de session sont valides pour la durée de la session en cours. Lorsqu'une nouvelle session JDBC est obtenue, le nom de schéma de la nouvelle session sera le nom de schéma par défaut.

SET SCRIPTFORMAT

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

Modifie le format du fichier script. Les formats BINARY et COMPRESSED sont légèrement plus rapides et plus compacts que le format par défaut TEXT.

Recommandé seulement pour les fichiers script de très grande taille

SET TABLE INDEX

```
SET TABLE nomTable INDEX 'index1rootPos index2rootPos ... ';
```

Cette commande n'est utilisée qu'en interne pour enregistrer la position des racines d'index dans le fichier .data. Elle n'apparaît que dans les fichiers script et ne devrait pas être utilisée directement.

SET TABLE READONLY

```
SET TABLE <nomTable> READONLY {TRUE | FALSE};
```

Spécifie que la table est en lecture seule.

SET TABLE SOURCE

```
SET TABLE <nomTable> SOURCE <fichier et options> [DESC];
```

Voir le chapitre Tables texte pour plus de détails.

Cette commande est utilisée exclusivement avec des tables TEXT pour indiquer quel fichier est utilisé pour enregistrer les données. Le qualificateur optionnel DESC a comme effet que le fichier est indexé depuis la fin et ouvert en lecture seule. L'argument <fichier et options> est une chaîne de caractères entourée de guillemets doubles qui est constituée de :

```
<fichier et options>::= <guillemet double> <cheminDeFichier>
[<point-virgule> <option>...] <guillemet double>
```

Exemple:

```
SET TABLE maTable SOURCE "monFichier;fs=|;vs=.;lvs=~"
```

Propriétés prises en charge:

Propriété	Description
quoted = { true | false }	La valeur par défaut est true. Si la valeur est false, les guillemets doubles sont traités comme des caractères normaux.
all_quoted = { true | false }	La valeur par défaut est false. Si la valeur est true, des guillemets doubles encadreront tous les champs.

encoding = <nomEncodage>	Encodage des caractères pour les champs texte ou caractère, par exemple : encoding=UTF-8.
ignore_first = { true | false }	La valeur par défaut est false. Si la valeur est true, la première ligne du fichier est ignorée.
cache_scale= <valeurNumérique>	Exposant pour déterminer le nombre de lignes du fichier texte mis en cache. La valeur par défaut est 8, équivalent à environ 800 lignes.
cache_size_scale = <valeurNumérique>	Exposant pour déterminer la taille moyenne de chaque ligne en cache. La valeur par défaut est 8, équivalent à 256 octets par ligne.
fs = <caractèreSansGuillemets>	Séparateur de champs
vs = <caractèreSansGuillemets>	Séparateur de champs Varchar
lvs = <caractèreSansGuillemets>	Séparateur de champs LongVarchar

Indicateurs spéciaux pour les séparateurs de Tables texte Hsqldb

Indicateur	Description
\semi	point-virgule
\quote	guillemet (simple)
\space	Caractère espace
\apos	apostrophe
\n	Saut de ligne - Utilisé comme ancre de fin (comme $ dans les expressions régulières)
\r	Retour chariot
\t	Tabulation
****	Antislash
\u####	Caractère Unicode exprimé en hexadecimal

Nécessite les privilèges d'administrateur.

SET WRITE_DELAY

```
SET WRITE_DELAY {{ TRUE | FALSE } | <secondes> | <millisecondes> MILLIS};
```

Permet de contrôler à quelle fréquence le fichier .log est synchronisé. Lorsque

l'on attribue la valeur FALSE ou 0 à WRITE_DELAY, le synchronisation a lieu immédiatement à chaque COMMIT. Lorsque l'on attribue la valeur TRUE à WRITE_DELAY, la synchronisation a lieu toutes les 20 secondes (qui est la valeur par défaut). Vous pouvez indiquer une autre durée à la place.

Le rôle de cette commande est de contrôler le volume de données perdues en cas de blocage système total. Une durée d'une seconde signifie que, au plus, les données inscrites sur le disque durant la seconde précédant le bloquage seront perdues. Toutes les données écrites auparavant auront été synchronisées et devraient être récupérables.

Un WRITE_DELAY de 0 a un impact sur les performances dans les situations de forte charge puisque le moteur de base de données doit attendre que les opérations soient effectuées par le système de fichiers.

Pour éviter ce problème, vous pouvez utiliser un WRITE_DELAY de 10 millisecondes. Dans la pratique, un WRITE_DELAY de 100 millisecondes permet une fiabilité supérieure à 99.9999% pour une moyenne d'un bloquage système par jour, et une fiabilité supérieure à 99.99999% pour une moyenne d'un bloquage système par semaine

À chaque fois que l'instruction SET_WRITE_DELAY est exécutée, une synchronisation est immédiatement réalisée.

Nécessite les privilèges d'administrateur.

SHUTDOWN

```
SHUTDOWN [IMMEDIATELY | COMPACT | SCRIPT];
```

Ferme la base de données.

Déclinaisons de la commande SHUTDOWN

SHUTDOWN
: Exécute un CHECKPOINT pour créer un nouveau fichier .script de taille minimale et qui contient les données des tables MEMORY seulement. Réalise ensuite une sauvegarde du fichier .data contenant les données des tables CACHED au format zip vers le fichier .backup et ferme la base de données.

SHUTDOWN IMMEDIATELY
: Ferme simplement les fichiers de la base de données (comme lorsque le processus Java de la base de données est interrompu). Cette commande est utilisée pour tester le mécanisme de récupération de données. Cette commande ne doit pas être utilisée comme méthode standard pour fermer la base de données.

SHUTDOWN COMPACT
: Crée un nouveau fichier .script qui contient les données de toutes les

tables, y-compris les tables CACHED et TEXT. Les fichiers des tables TEXT et le fichier .data sont ensuite supprimés, puis ré-écrits. Ensuite, le fichier .data est sauvegardé de la même manière que lors d'un SHUTDOWN normal. Cette opération réduit tous les fichiers à leur taille minimale.

SHUTDOWN SCRIPT

Similaire à SHUTDOWN COMPACT, mais après l'écriture du fichier .script et la suppression des fichiers existants, les fichiers .data et ceux des tables TEXT ne sont pas recréés. Après l'exécution de SHUTDOWN SCRIPT, seuls restent les fichiers .script et .properties. Au démarrage suivant de la base de données, ces fichiers sont traités et les fichiers .data et .backup sont créés. Dans la pratique, cette commande exécute une partie des tâches d'une instruction SHUTDOWN COMPACT, la suite étant exécutée automatiquement au démarrage suivant de la base de données.

Cette commande génère un fichier script complet de la base de données qui peut ensuite être modifié si besoin est avant le démarrage suivant.

Seul un administrateur peut exécuter l'instruction SHUTDOWN.

UPDATE

```
UPDATE table SET colonne = Expression [, ...] [WHERE Expression];
```

Modifie des données d'une table de la base de données.

Exemples de commande à saisir en SQL direct :

```
UPDATE "Clients"
SET "Nom" = (SELECT "Nom" FROM "AjoutClients" WHERE "AjoutClients"."ID" = "Clients"."ID")
```

```
UPDATE "tbl-articles"
SET "PrixHT" = (SELECT "Prixnet" FROM "tbl-tarifs" WHERE "Reffrs" = "tbl-articles"."reffrs") WHERE EX
(SELECT "Prixnet" FROM "tbl-tarifs" WHERE "tbl-articles"."reffrs" = "Reffrs")
```

ou voir |ici| (https://forum.openoffice.org/fr/forum/viewtopic.php?t=43805&p=239227#p239222) l'exemple.

Voir aussi: SELECT[2], INSERT, DELETE.

Nomenclature des objets de schéma

Les objets de schéma sont des objets de la base de données qui sont toujours orientés vers un schéma spécifique. Chaque schéma a un nom d'espace. Il peut y avoir de multiples objets de schéma du même nom, chacun dans un nom d'espace d'un schéma différent. Un objet de schéma particulier peut pratiquement toujours être identifié de façon unique en utilisant la notation NomDeSchéma.NomDObjet . Tous les objets de base de données HSQLDB sont

des objets de schéma, exceptés les suivants :

Users

Roles

Store Procedure Java Classes

HSQL Aliases

Nos déclencheurs actuels basés sur les classes Java ne sont pas pleinement des objets de schéma. Toutefois, nous sommes en train d'implémenter des déclencheurs conformes au SQL qui vont englober nos déclencheurs basés sur les classes Java. Quand ce travail sera terminé, les déclencheurs HSQLDB seront des objets de schéma.

Les séquences sont des objets de schéma dont les permissions de création et de suppression sont gouvernées par les autorisations du schéma (comme décrit ci-après). Cependant les commandes GRANT et REVOKE ne fonctionnent pas actuellement pour les séquences. Dans une version future d'HSQLDB, les GRANTs et REVOKEs de séquences fonctionneront d'une façon similaire aux commandes actuelles GRANT et REVOKE pour l'accès aux tables.

La plupart du temps, vous n'avez pas besoin de spécifier le schéma pour l'objet de schéma désiré, parce que le schéma implicite est généralement le seul qui puisse être utilisé. Par exemple, lors de la création d'un index, le schéma cible sera par défaut celui de la table qui est la cible de l'index. Les contraintes nommées en sont un exemple extrême. Il n'est jamais besoin de spécifier un nom de schéma pour une contrainte, depuis que les noms de contrainte sont uniquement définis dans une commande CREATE ou ALTER TABLE, et que le schéma doit être celui de la table cible. Si le schéma implicite n'est pas déterminé par un objet relatif, la valeur par défaut vient des réglages de schéma de votre session JDBC en cours. La valeur du schéma de la session sera votre nom d'utilisateur du schéma initial, ou ce que vous avez défini la dernière fois dans votre session JDBC en cours avec la commande SET SCHEMA. (Votre schéma initial est "PUBLIC" sauf s'il a été changé avec ALTER USER SET INITIAL SCHEMA ou la commande).

En plus de la portée du nom d'espace, il y a les aspects de permission du schéma de l'objet de la base de données. L'autorisation d'un schéma est un rôle ou bien l'utilisateur qui est simplement le propriétaire du schéma. Seulement un utilisateur avec le rôle DBA (administrateur de base de données) ou le propriétaire d'un schéma peuvent créer des objets, ou modifier le DDL (langage de définition des données) des objets, dans le nom d'espace de ce schéma. Une autorisation de schéma / un propriétaire peut être un rôle ou un utilisateur (même un rôle sans membres). Les deux schémas automatiquement générés quand une base de données est créée appartiennent au rôle DBA.

Une implication importante dans les objets de base de données qui appartiennent au propriétaire du schéma est que, si un utilisateur de base de données non DBA doit avoir la permission de créer tout objet de la base de données, ces objets ont les droits de propriétés du schéma. Pour permettre à un utilisateur de créer (ou modifier le DDL) des objets dans leur propre schéma personnel, vous devez créer un nouveau schéma où cet utilisateur est autorisé. Pour permettre à un utilisateur non DBA de partager les droits de création et de DDL dans un schéma, vous devez créer ce schéma muni d'un rôle comme étant l'autorisation, puis accorder ce rôle à tous les utilisateurs souhaités.

Le schéma INFORMATION_SCHEMA est un schéma défini par le système qui contient les tables système de la base de données. Ce schéma est en lecture seule. Lors de la création d'une base de données, un schéma nommé PUBLIC est automatiquement créé comme le schéma par défaut. Ce schéma a l'autorisation DBA. Vous pouvez changer le nom de ce schéma. Si tous les schémas non systèmes sont supprimés d'une base de données, un schéma PUBLIC vide est à nouveau créé. Et donc chaque base de données a toujours au moins un schéma non système.

Types de données

Tableau 9.1: Types de données. Les différents types sur une même ligne sont équivalents.

Nom	Etendue	Type Java
INT	comme le type Java	java.lang.Integer
FLOAT	comme le type Java	java.lang.Double
VARCHAR	comme Integer.MAXVALUE	java.lang.String
VARCHAR_IGNORECASE	comme Integer.MAXVALUE	java.lang.String
CHARACTER	comme Integer.MAXVALUE	java.lang.String
LONGVARCHAR	comme Integer.MAXVALUE	java.lang.String
DATE	comme le type Java	java.sql.Date
TIME	comme le type Java	java.sql.Time
DATETIME	comme le type Java	java.sql.Timestamp
DECIMAL	pas de limite	java.math.BigDecimal
NUMERIC	pas de limite	java.math.BigDecimal
BIT	comme le type Java	java.lang.Boolean
TINYINT	comme le type Java	java.lang.Byte
SMALLINT	comme le type Java	java.lang.Short
BIGINT	comme le type Java	java.lang.Long
REAL	comme le type Java	java.lang.Double
BINARY	comme Integer.MAXVALUE	byte[]
VARBINARY	comme Integer.MAXVALUE	byte[]
LONGVARBINARY	comme Integer.MAXVALUE	byte[]
OBJECT	comme Integer.MAXVALUE	java.lang.Object

Les noms en majuscules sont les noms des types de données définis par la norme SQL ou utilisés couramment par des SGBDR. Étendue indique la taille maximum de l'objet qui peut être enregistré. Lorsque la mention Integer.MAXVALUE est indiquée, cela est la limite théorique et dans la pratique la taille maximale d'un objet VARCHAR ou BINARY qui peut être enregistré est conditionnée par la quantité de mémoire disponible. Des objets d'une taille pouvant atteindre un mega-octet ont été utilisé avec succès dans

des bases de données en production.

La correspondance recommandée pour le type de données JDBC FLOAT est le type Java "double". En raison de la confusion potentielle, il est recommandé d'utiliser le type DOUBLE à la place de FLOAT.

VARCHAR_IGNORECASE est un type de VARCHAR particulier insensible à la casse. Ce type n'est pas portable.

Dans les instructions de définition de tables, HSQLDB accepte des qualificateurs de taille, précision et échelle pour certains types : CHAR(s), VARCHAR(s), DOUBLE(p), NUMERIC(p), DECIMAL(p,s) et TIMESTAMP(p).

TIMESTAMP(p) n'accepte que 0 ou 6 comme précision. Zéro indique **no subsecond part**. Sans précision, la valeur par défaut est 6.

Par défaut, la précision ou l'échelle indiquées pour une colonne sont ignorées par le moteur de base de données. Ce sont les valeurs de types Java correspondant qui sont utilisées, qui dans le cas de DECIMAL sont une précision et une échelle illimitées. Si une taille est spécifiée, elle est sauvegardée dans la définition de la base de données mais n'est pas appliquée par défaut. Après la création de la base de données, avant d'ajouter des données, vous pouvez attribuer une valeur à une propriété de la base pour appliquer le respect des tailles :

```
SET PROPERTY "sql.enforce_strict_size" true
```

Cela imposera la précision et l'échelle spécifiées et rajoutera des espaces aux champs CHAR pour obtenir la taille spécifiée. Cela est conforme au standard SQL en retournant une exception si l'on tente d'insérer une chaîne de caractères dont la taille est supérieure à la taille maximale. Cela imposera également aux valeurs DECIMAL le respect de la précision et de l'échelle spécifiées.

Par défaut, les colonnes CHAR, VARCHAR et LONGVARCHAR sont comparées et triées conformément au standard POSIX. Voir la section SET DATABASE COLLATION pour modifier ce comportement. La propriété sql.compare_in_locale n'est plus supportée. À la place, vous pouvez spécifier une collation à utiliser pour les comparaisons de caractères.

Les colonnes de type OTHER ou OBJECT contiennent une forme sérialisée d'objet Java au format binaire. Pour insérer ou modifier des données dans ces types de colonnes, une chaîne au format binaire doit être utilisée. L'utilisation de PreparedStatement avec JDBC automatise cette transformation.

Lignes de commentaire SQL

```
-- Ligne de commentaire style SQL
```

```
// Ligne de commentaire style Java
/* Ligne de commentaire style C */
```

Tous ces types de commentaires sont ignorés par la base de données.

Procédures stockées / Fonctions.

Les procédures stockées sont des fonctions Java statiques qui sont appelées directement, ou par un alias, dans une instruction SQL. L'appel d'une d'une fonction Java (directement ou par un alias) nécessite que la classe Java soit accessible par la base de données. La syntaxe est :

```
"java.lang.Math.sqrt"(2.0)
```

Cela signifie que le package doit être fourni, le nom doit être écrit en un seul mot et entouré de guillemets doubles car sinon il est converti en majuscules et est introuvable. Vous pouvez créer un alias avec la commande

```
CREATE ALIAS: CREATE ALIAS SQRT FOR "java.lang.Math.sqrt";
```

Lorsqu'un alias a été défini, vous pouvez appeler la fonction par son alias :

```
SELECT SQRT(A) , B FROM MYTABLE;
```

Seule les méthodes Java statiques peuvent être utilisées comme procédures stockées. Si dans la même classe il-y-a des méthodes surchargées comportant le même nombre d'arguments, c'est la première rencontrée par le programme qui sera utilisée. Si vous souhaitez utiliser des méthodes de la bibliothèque Java, il est recommandé de créer vos propres classes avec des méthodes statiques qui encapsulent les méthodes de la bibliothèque Java. cela vous permettra de contrôler quelle signature de méthode est utilisée pour appeler chaque méthode de la bibliothèque Java.

Les fonctions intégrées et procédures stockées

Numérique

(Numerical built-in Functions / Stored Procedures)

Fonction	Description
ABS(d)	retourne la valeur absolue d'une valeur de type double
ACOS(d)	retourne l'arc cosinus d'un angle ASIN(d) retourne l'arc sinus d'un angle
ATAN(d)	retourne l'arc tangente d'un angle

ATAN2(a,b)	retourne l'arc tangente de a/b
BITAND(a,b)	retourne a & b
BITOR(a,b)	b
CEILING(d)	retourne le plus petit entier qui n'est pas strictement inférieur à d
COS(d)	retourne le cosinus d'un angle
OT(d)	retourne la cotangente d'un angle
DEGREES(d)	convertit les radians en degrés
EXP(d)	retourne e (2.718...) puissance d
FLOOR(d)	retourne le plus grand entier qui n'est pas strictement supérieur à d
LOG(d)	retourne le logarithme naturel de d (base e)
LOG10(d)	retourne le logarithme de d (base 10)
MOD(a,b)	retourne a modulo b PI() retourne pi (3.1415...)
POWER(a,b)	retourne a puissance b
RADIANS(d)	convertit les degrés en radian
RAND()	retourne un nombre aléatoire supérieur ou égal à 0 et strictement inférieur à 1
ROUND(a,b)	arrondit a à b chiffres après la virgule
ROUNDMAGIC(d)	résout les problèmes d'arrondi tels que 3.11-3.1-0.01
SIGN(d)	retourne -1 si d est strictement inférieur à 0, 0 si d est égal à 0, 1 si d est strictement supérieur à 0
SIN(d)	retourne le sinus d'un angle
SQRT(d)	retourne la racine carrée
TAN(A)	retourne la tangente d'un angle
TRUNCATE(a,b)	tronque a à b chiffres après la virgule

- Supprimer une ligne sur deux avec MOD (http://user.services.openoffice.org/fr/forum/ftopic1543.html#p127907)
- Arrondi au demi point supérieur avec ROUND et TRUNCATE (http://user.services.openoffice.org/fr/forum/ftopic19892.html)

Chaîne de caractères

(Numerical built-in Functions / Stored Procedures)

Fonction	Description
ASCII(s)	retourne le code ASCII du premier caractère de s
BIT_LENGTH(str)	retourne la longueur de la chaîne en bits
CHAR(c)	retourne le caractère dont le code ASCII est c
CHAR_LENGTH(str)	retourne la longueur de la chaîne en caractères
CONCAT(str1,str2)	retourne str1 + str2
DIFFERENCE(s1,s2)	returns the difference between the sound of s1 and s2
HEXTORAW(s1)	retourne les caractères Unicode correspondants aux valeurs hexadecimales de s1
INSERT(s,debut,long,s2)	retourne une chaîne de caractères où 'long' caractères à partir de 'debut' ont été remplacés par s2
LCASE(s)	convertit s en minuscules
LEFT(s,nbre)	retourne les 'nbre' premiers caractères de s. Nécessite des guillemets doubles, utilisez plutôt SUBSTRING()
LENGTH(s)	retourne le nombre de caractères de s
LOCATE(rech,s,[debut])	retourne l'index de la première occurence de (1=premier caractère, O = non trouvé) de 'rech' dans 's', à partir de 'debut'
LTRIM(s)	supprime tous les espaces présents au début de s
OCTET_LENGTH(str)	retourne la longueur de str en octets, soit le double du nombre de caractères
RAWTOHEX(s1)	retourne la valeur Unicode des caractères de s1 en hexadecimal
REPEAT(s,nbre)	retourne s répétée 'nbre' fois.
REPLACE(s,remplace,s2)	remplace toutes les occurences de 'remplace' dans 's' par 's2'
RIGHT(s,nbre)	retourne les 'nbre' derniers caractères de s
RTRIM(s)	supprime tous les espaces présents à la fin de s
SOUNDEX(s)	retourne un code de quatre caractères représentant le son de s
SPACE(nbre)	retourne une chaîne de caractères constituée de 'nbre' espaces.

SUBSTR(s,debut[,longueur])	alias pour substring SUBSTRING(s,debut[,longueur]) retourne la sous-chaîne de caractères de s débutant à 'debut' (1 = premier caractère) et de longueur 'longueur'
UCASE(s)	convertit s en majuscules
LOWER(s)	convertit s en minuscules
UPPER(s)	convertit s en majuscules

- Mettre le Nom et prénom dans un même champ avec CONCAT (http://user.services.openoffice.org/fr/forum/ftopic16978.html)
- Dédoubler une colonne avec LOCATE SUBSTRING et LEFT (http://user.services.openoffice.org/fr/forum/ftopic31183.html)
- Remplacer du contenu dans une table avec REPLACE (http://user.services.openoffice.org/fr/forum/ftopic30091.html)
- Équivalent de la fonction NomPropre de Calc avec UCASE et UPPER (http://user.services.openoffice.org/fr/forum /ftopic24498.html)
- Retourner le NOM (majuscules) en Nom (initiale en majuscule, suivantes en minuscules) avec UCASE, LCASE, LENGTH, LEFT et RIGHT (http://user.services.openoffice.org/fr/forum /ftopic52053.html)

Date/Heure

(Numerical built-in Functions / Stored Procedures)

Fonction	Description
CURDATE()	retourne la date courante
CURTIME()	retourne l'heure courante
DATEDIFF(chaîne, dateheure1, dateheure2)	retourne le nombre d'unités de temps écoulée entre 'dateheure1' et 'dateheure2'. 'chaîne' indique l'unité de temps et peux avoir les valeurs suivantes : 'ms'='millisecond' (millisecondes), 'ss'='second' (secondes),'mi'='minute' (minute),'hh'='hour' (heure), 'dd'='day' (jour), 'mm'='month' (mois), 'yy' = 'year' (année). Les deux formes courte et longue de 'chaîne' peuvent être utilisées.
DAYNAME(date)	retourne le nom du jour (en Anglais)
DAYOFMONTH(date)	retourne le jour du mois (1-31)

DAYOFWEEK(date)	retourne le jour de la semaine (1 signifie Dimanche)
DAYOFYEAR(date)	retourne le jour de l'année (1-366)
HOUR(time)	retourne l'heure (0-23)
MINUTE(time)	retourne les minutes (0-59)
MONTH(date)	retourne le mois (1-12)
MONTHNAME(date)	retourne le nom du mois (en Anglais)
NOW()	retourne la date et l'heure en cours, comme un timestamp. - Utilisez plutôt CURRENT_TIMESTAMP
QUARTER(date)	retourne le trimestre (1-4)
SECOND(time)	retourne les secondes (0-59)
WEEK(date)	retourne le numéro de semaine dans l'année (1-53). Attention, pour hsqldb les semaines commencent le Dimanche
YEAR(date)	retourne l'année
CURRENT_DATE	retourne la date courante
CURRENT_TIME	retourne l'heure courante
CURRENT_TIMESTAMP	retourne un horodatage (format date et heure)

- Tri avec fonctions MONTH et YEAR (http://user.services.openoffice.org/fr/forum/ftopic15656.html)
- Données de la journée avec CURRENT_DATE (http://user.services.openoffice.org/fr/forum/ftopic2762.html)
- Calcul d'âge avec DAY MONTH YEAR et CURRENT_DATE (http://user.services.openoffice.org/fr/forum/ftopic4354.html)
- Obtenir la date du jour dans un formulaire avec CURRENT_TIMESTAMP (http://user.services.openoffice.org/fr/forum/ftopic9183.html)
- Regroupement sur le mois avec MONTHNAME (http://user.services.openoffice.org/fr/forum/ptopic9543.html)

Connexions

(Numerical built-in Functions / Stored Procedures)

Fonction	Description

DATABASE()	retourne le nom complet (y-compris le chemin de fichier) de la base de données de la connexion courante
USER()	retourne le nom d'utilisateur de la connexion courante
CURRENT_USER fonction SQL standard.	Retourne le nom d'utilisateur de la connexion courante
IDENTITY()	retourne la dernière valeur d'IDENTITY insérée par la connexion courante

- Liaison dynamique d'un fichier CSV avec IDENTITY (http://user.services.openoffice.org/fr/forum/ftopic13620.html)

Logiques (Système)

(Numerical built-in Functions / Stored Procedures)

Fonction	Description
IFNULL (exp,valeur)	retourne 'valeur' si 'exp' est null, sinon retourne 'exp'. Utilisez plutôt COALESCE() à la place.
CASEWHEN (exp,v1,v2)	Si 'exp' vaut true, 'v1' est retournée, sinon 'v2'. Utilisez plutôt CASE WHEN à la place.
CONVERT (term,type)	convertit term vers un autre type de données
CAST (term AS type)	convertit term vers un autre type de données
COALESCE (expr1,expr2,expr3,...)	retourne expr1 si expr1 n'est pas null, sinon évalue expr2 et retourne expr2 si expr2 n'est pas null, et ainsi de suite...
NULLIF (v1,v2)	retourne null si v1 est égal à v2, sinon retourne v1
CASE v1 WHEN...	CASE v1 WHEN v2 THEN v3 [ELSE v4] END (END est nécessaire au fonctionnement de CASE WHEN) retourne v3 si v1 est égal à v2, sinon retourne v4, ou null s'il n'y-a-pas de clause ELSE
CASE WHEN...	CASE WHEN expr1 THEN v1[WHEN expr2 THEN v2] [ELSE v4] END retourne v1 lorsque expr1 vaut true, optionellement répété pour d'autres cas, sinon retourne v4, ou null s'il n'y-a-pas de clause ELSE

EXTRACT	MONTH | DAY | HOUR | MINUTE | SECOND} FROM <valeurDateHeure>)
POSITION (... IN ..)	POSITION(<chaîneDeCaractères> IN <chaîne DeCaractères>) retourne la position (numérotée à partir de 1) de la première chaîne de caractère si c'est une sous-chaîne de la deuxième chaîne de caractères. Sinon, retourne 0
SUBSTRING (... FROM ... FOR ...)	SUBSTRING(<chaîneDeCaractères> FROM <expressionNumérique1> [FOR <expressionNumérique2>]) retourne la sous-chaîne <chaîneDeCaractères> depuis la position <expressionNumérique1> [pour <expressionNumérique2> caractère(s)])
TRIM (LEADING ... FROM ...)	TRAILING | BOTH}] FROM <chaîneDeCaractères>)

Voir aussi : CALL, CREATE ALIAS.

- Afficher les âges à différentes dates avec CASE WHEN et CAST (http://user.services.openoffice.org/fr/forum/ftopic15843.html)
- Préciser un type de paiement avec CASE WHEN (http://user.services.openoffice.org/fr/forum/ftopic23697.html)
- Déterminer la date d'un anniversaire avec CAST (http://user.services.openoffice.org/fr/forum/ftopic28605.html)
- Concaténer même quand un champ est vierge avec COALESCE (http://user.services.openoffice.org/fr/forum/ftopic13620.html)
- Ajouter un nombre à une date avec CAST et CONVERT (http://user.services.openoffice.org/fr/forum /ftopic1543.html#p81847)
- Modifier l'ordre des caractères avec SUBSTRING (http://user.services.openoffice.org/fr/forum/ftopic3720.html)

Expression SQL

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

Composants des expressions SQL

condition

```
{ valeur [|| valeur]
| valeur { = | < | <= | > | >= | <>
| != } valeur
| valeur IS [NOT] NULL
| EXISTS(instructionDeSélection)
| valeur BETWEEN valeur AND valeur
| valeur [NOT] IN ( {valeur [, ...]
| instructionDeSélection } )
| valeur [NOT] LIKE valeur [ESCAPE] valeur }
```

valeur

```
[+ | -] { terme [{ + | - | * | / | || } terme]
| ( condition ) | fonction ( [paramètre] [,...] )
| instructionDeSélection retournant une seule valeur
| {ANY|ALL} (instructionDeSélection retournant une seule colonne)
```

terme

```
{ 'chaîneDeCaractères' | nombre | virguleFlottante
| [table.]colonne | TRUE | FALSE | NULL }
```

séquence

```
NEXT VALUE FOR <séquence>
```

HSQLDB ne respecte pas actuellement la proposition de règles SQL 200n dans laquelle les valeurs générées par une séquence peuvent être utilisées. En général, ces valeurs peuvent être utilisées dans les instructions INSERT et UPDATE, mais pas dans les instructions CASE, dans les clauses ORDER BY, dans les conditions de recherche, dans les fonctions d'aggrégation ou dans les requêtes groupées.

chaînes de caractères

Les chaînes de caractères dans HSQLDB sont des chaînes Unicode. Une chaîne débute et finit par un guillemet simple '. Dans une chaîne qui débute par un guillemet simple, utiliser deux guillemets simple *pour créer un nguillemet simple.*

La concaténation des chaînes de caractères doit être réalisée de préférence avec l'opérateur standard SQL

---

nom

Le jeu de caractères pour les identifiants (les noms) dans HSQLDB est l'Unicode.

Un identifiant non entouré de guillemets doubles doit commencer par une lettre et être suivi de lettres ou de nombres ASCII. Lorsqu'une instruction SQL est générée, tous les caractères en minuscule d'un identifiant non entouré de guillemets doubles sont convertis en majuscules. Pour cette raison, les noms qui ne sont pas entourés de guillemets doubles sont en fait ALL UPPERCASE lorsqu'ils sont utilisés dans une instruction SQL. Cela a une conséquence importante lors de l'accès aux noms de colonnes par l'intermédiaire des méta-données JDBC DatabaseMetaData : la forme interne, qui est ALL UPPERCASE, doit être utilisée si le nom de colonne n'était pas entouré de guillemets doubles dans l'instruction CREATE TABLE.

Les identifiants entourés de guillemets doubles peuvent être utilisés comme noms (pour les tables, les colonnes, les contraintes ou les index). Les identifiants entre guillemets doubles commencent et finissent par un guillemet double ". Un identifiant entre guillemets doubles peut contenir n'importe quel caractère Unicode, y-compris un espace. Dans un identifiant entre guillemets doubles, utiliser "" (deux guillemets doubles) pour créer un guillemet double ". Avec les identifiants entre guillemets doubles, il est possible de créer des noms de tables ou de colonnes comportant des majuscules et des minuscules. Exemple :

```
CREATE TABLE "Adresse" ("Nr" INTEGER,"Nom" VARCHAR);
SELECT "Nr", "Nom" FROM "Adresse";
```

L'identifiant entre guillemets doubles équivalent à un identifiant non entouré de guillemets peut être utilisé en le convertissant en majuscules. Par exemple, si le nom d'une table est défini comme Adresse2 (sans guillemets), on peut y faire référence par sa forme entre guillemets doubles "ADRESSE2", ainsi que par adresse2, aDReSSe2 et ADRESSE2. Attention à ne pas confondre les

identifiants entre guillemets doubles avec les chaînes de caractères SQL.

Les guillemets doubles peuvent parfois être utilisés pour des identifiants, des alias ou des fonctions lorsqu'il-y-a une ambiguïté :

```
SELECT COUNT(*) "COUNT" FROM MYTABLE;
SELECT "LEFT"(COL1, 2) FROM MYTABLE;
```

Bien que HSQLDB 1.8.0 n'impose pas l'utilisation exclusive de caractères ASCII pour les identifiants sans guillemets, l'utilisation de caractères non ASCII dans ces identifiants n'est pas conforme à la norme SQL. L'utilisation de caractères accentués (ou de caractères Unicode étendu) dans des identifiants non entourés de guillemets pourraît provoquer des problèmes lors du portage vers différentes localisations du JRE. Puisque ce sont des méthodes Java natives qui sont utilisées pour convertir en majuscules les identifiants non entourés de guillemets doubles, le résultat peut être différent suivant la localisation Java. Il est donc recommandé de n'utiliser des caractères accentués que dans les identifiants entre guillemets doubles.

Lors de l'utilisation de méthodes JDBC DatabaseMetaData qui utilisent des identifiants de tables, de colonnes ou d'index comme arguments, utilisez les noms tels qu'ils sont enregistrés dans la base de données. Avec ces méthodes, les identifiants sans guillemets doubles doivent être utilisés en majuscules pour obtenir un résultat correct. Les identifiants entre guillemets doubles doivent être utilisés avec les mêmes combinaisons de casse que lors de leur définition. Aucun guillemet ne doit être ajouté autour du nom. Les méthodes JDBC qui retournent un result set contenant de tels identifiants retournent les identifiants sans guillemets en majuscules et les identifiants entre guillemets doubles dans les même combinaisons de casse que celle avec laquelle ils ont été enregistrés dans la base de données.

Notez également que les méthodes JDBC getXXX(String nomColonne) interprètent les noms de colonnes de manière insensible à la casse. C'est une fonctionalité de JDBC qui n'est pas spécifique à HSQLDB.

Mot de passe

Les mots de passe doivent être entourés de guillemets doubles. Les mots de passe sont insensibles à la casse uniquement pour des

raisons de compatibilité avec des versions antérieures. Cela peut changer dans le futur.

valeurs

- Une DATE sous forme littérale commence et finit par ' (guillemet simple), le format est aaaa-mm-jj (voir java.sql.Date.
- Un TIME sous forme littérale commence et finit par ' (guillemet simple), le format est hh:mm:ss (voir java.sql.Time).
- Un TIMESTAMP ou DATETIME sous forme littérale commence et finit par ' (guillemet simple),le format est aaaa-mm-jj hh:mm:ss.SSSSSSSSS (voir java.sql.Timestamp).

Lorsque des valeurs par défaut sont indiquées pour des colonnes date / heure dans une instruction CREATE TABLE, ou lorsque des instructions SELECT, INSERT, UPDATE sont exécutées, les fonctions SQL NOW, SYSDATE, TODAY, CURRENT_TIMESTAMP, CURRENT_TIME et CURRENT_DATE (insensibles à la casse) peuvent être utilisées. NOW est utilisé pour les colonnes de type TIME et TIME_STAMP, TODAY est utilisé pour les colonnes de type DATE. Les différentes variantes date et heure de CURRENT_* sont des versions conformes à la norme SQL et doivent être utilisées de préférence aux autres. Exemple :

```
CREATE TABLE T(D DATE DEFAULT CURRENT_DATE);
CREATE TABLE T1(TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

Les données binaires commencent et finissent par ' (guillemets simples), le format est hexadécimal. Par exemple, '0004ff' représente 3 octets, 0, 4 puis 255.

Un nombre quelconque de commandes peuvent être combinées. Dans ce cas, le ';' (point virgule) doit être utilisé à la fin de chaque commande pour assurer l'intégrité des données, malgré le fait que le moteur puisse comprendre la fin des commandes et ne retourne pas d'erreurs lorsqu'on utilise pas le point virgule.

Exemples commentés

- Tri sur fonctions MONTH et YEAR (http://user.services.openoffice.org/fr/forum/ftopic15656.html)

- Exemple avec COALESCE : Concaténer même quand un champ est vierge (http://user.services.openoffice.org/fr/forum/ftopic13620.html)
- Exemple avec CASE WHEN, CAST : Requête pour âge à différentes dates (http://user.services.openoffice.org/fr/forum/ftopic15843.html)

Récupérée de « https://wiki.openoffice.org/w/index.php?title=FR/Documentation/HSQLDB_Guide/ch09&oldid=245794 »
Catégorie : FR/HSQLDB Guide

- Dernière modification de cette page le 11 octobre 2020 à 19:10.
- Content is available under ALv2 unless otherwise noted.