Le catalogue MySQL

par Cédric DUPREZ

Date de publication : 14/05/2008

Dernière mise à jour :

Le catalogue, ou dictionnaire de données, est présent dans MySQL depuis la version 5.0.2. Dans cet article, nous verrons les données proposées par le catalogue et des exemples d'utilisation pratique.

Préambule

Dans les requêtes présentées dans ce tutoriel, les crochets indiquent les portions de code SQL optionnelles.

I - Définitions

Le **catalogue** d'une **base de données** est lui-même une base contenant les métadonnées de cette première base, c'est-à-dire les informations décrivant sa structure.
Selon les **SGBD** (systèmes de gestion de bases de données), il peut y avoir un catalogue unique pour toutes les bases contenues, comme c'est le cas sous MySQL, ou alors un catalogue par base de données, comme sous Oracle.

Une **métadonnée** est une donnée qui permet de définir ou de décrire une autre donnée. Par exemple, le nom d'un champ, le type de ce champ, sa table d'appartenance sont autant de métadonnées contenues dans une base.

Or, ces informations ne sont pas accessibles directement par **SQL** lors de l'exécution d'une requête dans la base que l'on interroge.
Ainsi, les SGBD disposent de bases pour décrire les données qu'elles contiennent : les *dictionnaires de données* ou *catalogues*.

II - Implémentation dans MySQL

Depuis sa version 5.0.2, MySQL dispose d'un catalogue système pour l'ensemble des bases de données présentes dans le SGBD.
Le nom de cette base, accessible en lecture, est **INFORMATION_SCHEMA**.

En réalité, il s'agit d'une collection de **vues** système et non pas de tables physiques (aucun fichier ne leur est donc associé).
Pour se placer dans le catalogue, il suffit de taper la commande suivante :

```
USE INFORMATION_SCHEMA;
```

Sinon, vous pouvez également préfixer les noms des vues système par **INFORMATION_SCHEMA.**, comme présenté dans les requêtes de ce tutoriel.
Ce catalogue respecte les standard fixés par la norme SQL:2003.

Avant la version 5.0.2 de MySQL, la commande SHOW permettait d'obtenir les métadonnées d'une base. Cette commande a été maintenue (et même enrichie) dans les versions ultérieures, pour des raisons de compatibilités et pour ne pas frustrer les utilisateurs habitués à simplicité de la syntaxe de SHOW.
L'utilisation du catalogue se fait par des requêtes SQL « classiques » (SELECT ... FROM INFORMATION_SCHEMA... WHERE...), sans faire appel à une syntaxe particulière, et de manière très flexible (toute la syntaxe SQL ainsi que les fonctions de MySQL sont à votre disposition pour préciser vos critères de recherche dans le catalogue).
Il est possible, en une seule requête, grâce aux jointures, d'obtenir des informations sur différents objets (base de données, tables, index, colonnes...), là où, avec la commande SHOW, il aurait fallu faire plusieurs interrogations successives.

C'est pourquoi la documentation de MySQL recommande son utilisation, si la version que vous possédez le permet.

III - Droits sur le catalogue

Vous vous dites peut-être que le catalogue est encore une base de données réservée aux seuls administrateurs.
Et bien non ! Comme c'était d'ailleurs déjà le cas pour la commande SHOW, le catalogue donne accès aux métadonnées des objets (schémas, tables, colonnes...) sur lesquels vous avez des droits, et notamment des droits de lecture.

Dès lors que vous accédez en lecture à une table, vous pouvez accéder aux informations du catalogue la concernant.

Entrons à présent dans le vif du sujet : la présentation des différentes vues du dictionnaire de données.
Sans vouloir faire un jeu de mots facile, le but du présent tutoriel n'est pas de présenter un « catalogue » des informations accessibles par le dictionnaire de données. La documentation de MySQL est faite pour cela.
L'objectif ici est de mettre l'accent sur les informations les plus utiles, et les requêtes permettant d'y accéder, ainsi que sur les combinaisons entre les vues du catalogue pour ramener, en une seule requête, plusieurs informations pertinentes.

IV - Informations sur les bases de données

Sous MySQL, une base de données est créée par la commande :

Création d'une base de données
```
CREATE DATABASE nom_base;
```

Dans le catalogue, les bases de données sont nommées « schema ».
Les informations concernant ces bases sont donc décrites dans la vue système **SCHEMATA**.

Pour obtenir toutes les informations sur une base précise, utilisez la requête suivante :

```
SELECT *
FROM INFORMATION_SCHEMA.SCHEMATA
WHERE SCHEMA_NAME = 'nom_base';
```

Notez que si vous omettez la clause WHERE, vous obtiendrez toutes les bases de données présentes sur votre serveur et auxquelles vous avez accès, y compris d'ailleurs le dictionnaire.

Trois informations sont utiles dans cette vue système :

- SCHEMA_NAME : le nom de la base de données ;
- DEFAULT_CHARACTER_SET_NAME : le jeu de caractères par défaut ;
- DEFAULT_COLLATION_NAME : la collation par défaut ;

Toutes ces informations sont des paramètres fournis lors de la création de la base de données dans la commande CREATE DATABASE.
Le jeu de caractères et la collation par défaut peuvent être modifiés par la commande ALTER DATABASE.
Bien évidemment, vous pouvez ne cibler, dans la requête d'interrogation du catalogue, que certaines de ces informations. Dans ce cas, remplacez l'étoile (*) par le nom des informations qui vous intéressent.
Par exemple, la requête équivalente à l'ancienne commande SHOW DATABASES s'écrit :

Liste des bases de données
```
SELECT SCHEMA_NAME
FROM INFORMATION_SCHEMA.SCHEMATA;
```

V - Informations sur les tables

On crée une table par la commande SQL :

Création d'une table
```
CREATE TABLE nom_table
[(definitions,...)]
[options];
```

Les métadonnées des tables sont présentes dans la vue système **TABLES**.
Pour obtenir toutes les tables d'une base de données, utilisez la requête suivante :

```
SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'nom_base';
```

A chaque ligne renvoyée par la requête correspond une table, une vue ou une table temporaire de la base spécifiée dans la clause WHERE.
Les informations couramment utilisées dans cette vue système sont :

- TABLE_SCHEMA : le nom de la base de donnée ;
- TABLE_NAME : le nom de la table ;
- TABLE_TYPE : cette donnée peut prendre 3 valeurs :
  a TABLE : l'objet est une table ;
  b VIEW : l'objet est une vue. A noter que pour une vue, toutes les autres informations suivantes sont à NULL ;
  c TEMPORARY : l'objet est une table temporaire ;
- ENGINE : le moteur utilisé pour la table (InnoDB, MyISAM...) ;
- TABLE_ROWS : le nombre de lignes contenues dans la table ;
- DATA_LENGTH : taille des données dans la table, en **octets** ;
- MAX_DATA_LENGTH : taille maximale autorisée pour les données, en octets ;
- INDEX_LENGTH : taille des index, en octets ;
- AUTO_INCREMENT : indique si la table possède une colonne auto-incrémentée. Si c'est le cas, la donnée vaut 1, sinon elle vaut NULL ;
- CREATE_TIME : date et heure de création de la table ;
- TABLE_COLLATION : la collation par défaut sur la table. Il n'y a pas d'information sur le jeu de caractères par défaut, mais il peut se déduire de la collation.

Remarque : en version 5.0, un bug empêche l'affichage des tables temporaires...

Voyons donc dès à présent un premier exemple de requête avancée : comment obtenir les 10 tables les plus importantes en taille d'une base de données (avec leur taille respective en mega-octets) ?

Les 10 plus grosses tables d'une base

```
SELECT TABLE_NAME, ROUND((DATA_LENGTH + INDEX_LENGTH) / (1024 * 1024), 2) AS TAILLE
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'nom_base'
ORDER BY DATA_LENGTH + INDEX_LENGTH DESC
LIMIT 10;
```

VI - Informations sur les colonnes

Les métadonnées sur les colonnes (ou champs) sont contenues dans la vue système **COLUMNS**.
Pour obtenir toutes les colonnes d'une table dans une base de données, utilisez la requête suivante :

```
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'nom_base'
AND TABLE_NAME = 'nom_table'
ORDER BY ORDINAL_POSITION;
```

Cette requête classe toutes les colonnes de la table par ordre dans lequel elles ont été déclarées dans la table lors de sa création (sauf s'il y a eu des changements dans l'ordre des colonnes par la commande ALTER).
Les informations les plus utiles dans cette vue système sont les suivantes :

- TABLE_SCHEMA : le nom de la base de donnée ;
- TABLE_NAME : le nom de la table à laquelle appartient la colonne ;
- COLUMN_NAME : le nom de la colonne ;
- ORDINAL_POSITION : le rang de la colonne dans la table (ordre de présentation) ;
- COLUMN_DEFAULT : la valeur par défaut de la colonne ;
- IS_NULLABLE : indique si la colonne accepte la valeur NULL. Cette information (sous forme de chaîne de caractères) peut prendre 2 valeurs :
  a 'NO' : les valeurs NULL ne sont pas acceptées ;
  b 'YES' : les valeurs NULL sont possibles ;
- DATA_TYPE : le type de la colonne (varchar, char, int, date...) ;
- CHARACTER_MAXIMUM_LENGTH : le nombre maximum de caractères pour les champs de type chaîne de caractères (char, varchar, text...) ;
- CHARACTER_OCTET_LENGTH : cette valeur est généralement égale à la précédente, sauf pour les jeux de caractères multi-octets, comme l'UTF8 (dans ce cas précis, CHARACTER_OCTET_LENGTH = 3 * CHARACTER_MAXIMUM_LENGTH puisque l'UTF8 est codé sur 3 octets) ;
- NUMERIC_PRECISION : contient la précision (déclarée ou implicite) d'un champ de type numérique. Par exemple, la précision d'un champ de type DECIMAL (4, 2) est 4 ;
- NUMERIC_SCALE : contient l'échelle (déclarée ou implicite) d'un champ de type numérique. Par exemple, l'échelle d'un champ de type DECIMAL (4, 2) est 2 ;
- CHARACTER_SET_NAME : le jeu de caractère d'une colonne de type chaîne de caractères ;
- COLUMN_TYPE : le type de la colonne. Par rapport à la donnée DATA_TYPE, cette valeur est complétée du nombre de caractères maximum pour les chaînes de caractères, de la précision pour les entiers, de la précision et de l'échelle pour les nombres décimaux. Cette colonne précise également, pour les entiers, s'ils ont un signe ou non (UNSIGNED) ;
- COLUMN_KEY : donnée, quand elle est renseignée, indiquant que la colonne est indexée ; elle peut prendre 3 valeurs :
  a 'PRI' : la colonne fait partie de la clef primaire de la table ;
  b 'UNI' : la colonne fait partie d'une clef unique ;
  c 'MUL' : la colonne fait partie d'un index non unique (par exemple, parce qu'elle constitue une clef étrangère) ;
- EXTRA : dans cette colonne, on trouve notamment l'information AUTO_INCREMENT quand la colonne est auto-incrémentée ;
- PRIVILEGES : les privilèges dont dispose l'utilisateur qui a exécuté l'interrogation du dictionnaire sur la colonne ;
- COLUMN_COMMENT : un commentaire éventuellement renseigné lors de la création d'une table.

Grâce à ces informations, vous pouvez, par exemple, déterminer quelle est la clef primaire d'une table avec la requête suivante :

Clef primaire d'une table
```
SELECT COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'nom_base'
AND TABLE_NAME = 'nom_table'
AND COLUMN_KEY = 'PRI'
ORDER BY ORDINAL_POSITION;
```

Si cette requête renvoie plus d'une ligne, c'est que la clef primaire de la table est composée de plusieurs colonnes, ce qu'on appelle également une clef concaténée.

Vous pouvez également lister les colonnes indexées dans cette table :

Liste des colonnes indexées d'une table
```
SELECT COLUMN_NAME, COLUMN_KEY
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'nom_base'
```

Liste des colonnes indexées d'une table
```
AND TABLE_NAME = 'nom_table'
AND COLUMN_KEY IS NOT NULL
ORDER BY ORDINAL_POSITION;
```

Certaines de ces informations (index, privilèges...) peuvent être complétées par d'autres requêtes dans le catalogue, comme nous le verrons par la suite.

VII - Informations sur les index

C'est la vue système **STATISTICS** qui contient les métadonnées sur les index, même si nous venons de voir qu'une partie de cette information se retrouve également dans la vue système décrivant les colonnes.
Pour lister tous les index d'une table, utilisez la requête suivante :

```
SELECT *
FROM INFORMATION_SCHEMA.STATISTICS
WHERE TABLE_SCHEMA = 'nom_base'
AND TABLE_NAME = 'nom_table';
```

Les informations remarquables dans cette vue système sont :

- TABLE_SCHEMA : le nom de la base de donnée ;
- TABLE_NAME : le nom de la table à laquelle appartient l'index ;
- NON_UNIQUE : indicateur numérique de non unicité des valeurs de l'index :
  - 0 : les valeurs de l'index doivent être uniques. C'est le cas des index liés aux clefs primaires ou aux contraintes d'unicité ;
  - 1 : les valeurs de la colonne ne doivent pas nécessairement être uniques. C'est le cas des index liés aux clefs étrangères ;
- INDEX_NAME : le nom de l'index. Pour les index de clefs primaires, leur nom est PRIMARY ; pour les clefs étrangères, ils sont, par défaut, préfixés par 'FK_' ;
- SEQ_IN_INDEX : indique l'ordre de la colonne dans l'index. Dans le cas des clefs concaténées sur plusieurs champs, cette donnée indique l'ordre de la colonne dans l'index de la clef primaire. Dans ce dernier cas, l'unicité des valeurs porte donc sur l'ensemble de la clef (chaque colonne n'a pas nécessairement de contrainte d'unicité, mais la combinaison des différents champs est nécessairement unique) ;
- COLUMN_NAME : le nom de la colonne indexée ;
- CARDINALITY : le nombre de lignes indexées ;
- INDEX_TYPE : le type d'index, par exemple BTREE.

On peut constater que cette vue système apporte plus de renseignements sur les index que la vue système qui décrit les colonnes.

VIII - Informations sur les contraintes

Pour obtenir toutes les informations sur les contraintes d'une table, il faut utiliser la combinaison de deux nouvelles vues systèmes de métadonnées : **TABLE_CONSTRAINTS** et **KEY_COLUMN_USAGE**.
La première des deux vues fournit principalement le type de la contrainte, alors que la seconde donne les tables et colonnes qui utilisent cette contrainte.
L'exemple le plus complet du fonctionnement de ces deux vues système est celui des clefs étrangères.
Supposons que vous souhaitiez connaître les clefs étrangères d'une table. Nous avons vu jusque ici d'une clef étrangère entraînait l'indexation de la colonne concernée.
Mais le nom de l'index ne permet pas toujours de savoir qu'il s'agit d'une clef étrangère, le préfixe 'FK_' étant facultatif.
En revanche, la requête suivante donne la liste des clefs étrangères d'une table :

Liste des clefs étrangères d'une table
```
SELECT *
```

Liste des clefs étrangères d'une table
```
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_SCHEMA = 'nom_base'
AND TABLE_NAME = 'nom_table'
AND CONSTRAINT_TYPE = 'FOREIGN KEY';
```

Cette fois, aucune ambiguïté n'est possible : grâce au type de la contrainte, on sait qu'il s'agit bien de clefs étrangères.
Parmi les autres informations utiles de la vue système **TABLE_CONSTRAINTS**, on trouve :

- CONSTRAINT_SCHEMA : la base de donnée de la contrainte ;
- CONSTRAINT_NAME : le nom de la contrainte (qui est aussi le nom de l'index associé à la contrainte) ;
- TABLE_SCHEMA : le nom de la base de données de la table sur laquelle porte la contrainte (information identique à la base de la contrainte ci-dessus) ;
- TABLE_NAME : le nom de la table à laquelle appartient la contrainte ;
- CONSTRAINT_TYPE : le type de contrainte, qui peut prendre les valeurs suivantes :
  a 'PRIMARY KEY' : clef primaire ;
  b 'UNIQUE' : unicité des valeurs ;
  c 'FOREIGN KEY' : clefs étrangères.

Cependant, cette seule vue système ne permet pas de connaître la table et la colonne référencée par la clef étrangère. Elle ne permet pas non plus, dans le cas d'une clef primaire concaténée, de connaître les colonnes composant la clef (même si nous avons déjà vu comment résoudre cette dernière question).
C'est pourquoi il faut combiner cette vue avec **KEY_COLUMN_USAGE**.
On obtient alors, pour notre exemple de clef étrangère, la requête suivante :

Liste des informations sur les clefs étrangères
```
SELECT k.CONSTRAINT_SCHEMA, k.CONSTRAINT_NAME, k.TABLE_NAME, k.COLUMN_NAME
     , k.REFERENCED_TABLE_SCHEMA, k.REFERENCED_TABLE_NAME, k.REFERENCED_TABLE_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS k
INNER JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS c
       ON k.CONSTRAINT_SCHEMA = c.CONSTRAINT_SCHEMA AND k.CONSTRAINT_NAME = c.CONSTRAINT_NAME
WHERE c.CONSTRAINT_TYPE = 'FOREIGN KEY';
```

Les informations supplémentaires qui nous intéressent sont alors les suivantes :

- REFERENCED_TABLE_SCHEMA : base contenant la table référencée par la clef étrangère ;
- REFERENCED_TABLE_NAME : nom de la table référencée par la clef étrangère ;
- REFERENCED_COLUMN_NAME : nom de la colonne référencée.

Le catalogue nous donne ainsi toutes les informations sur les clefs étrangères.
Imaginez un peu la puissance de telles requêtes, quand on ajoute les informations sur les index liés à ces clefs étrangères. En une seule interrogation, vous collectez toutes les informations dont vous avez besoin, là où il fallait auparavant lancer deux à trois commandes SHOW indépendantes les unes des autres et tenter de recoller les morceaux « à la main »...

Terminons cette partie par un petit problème : comment trouver les clefs étrangères qui n'ont pas d'index ?
Par défaut, MySQL crée des index à la création des clefs étrangères.
Mais rien n'empêche ensuite la suppression de ces index.
Pour résoudre cette question, il faut l'utilisation conjointe de 3 des vues système que nous venons de voir, dans la requête suivante :

Liste des clefs étrangères non indexées
```
SELECT k.CONSTRAINT_SCHEMA, k.CONSTRAINT_NAME, k.TABLE_NAME, k.COLUMN_NAME
    , k.REFERENCED_TABLE_SCHEMA, k.REFERENCED_TABLE_NAME, k.REFERENCED_COLUMN_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS k
INNER JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS c
    ON k.CONSTRAINT_SCHEMA = c.CONSTRAINT_SCHEMA
```

Liste des clefs étrangères non indexées
```
    AND k.CONSTRAINT_NAME = c.CONSTRAINT_NAME
LEFT JOIN INFORMATION_SCHEMA.STATISTICS AS s
    ON k.CONSTRAINT_SCHEMA = s.TABLE_SCHEMA
    AND k.TABLE_NAME = s.TABLE_NAME
    AND k.COLUMN_NAME = s.COLUMN_NAME
WHERE c.CONSTRAINT_TYPE = 'FOREIGN KEY'
    AND s.INDEX_NAME is null;
```

La jointure externe (LEFT JOIN) sur STATISTICS couplée à la dernière condition permet de trouver les clefs étrangères pour lesquelles il n'y a pas d'index.

IX - Informations sur les procédures stockées, fonctions et déclencheurs

IX-A - Procédures stockées et fonctions

Le dictionnaire de données donne des informations sur les **procédures stockées** et fonctions (également regroupées sous le terme de routines), par la requête suivante :

```
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
```

Les métadonnées les plus pertinentes renvoyées par cette requête sont les suivantes :

- SPECIFIC_NAME : le nom de la routine ;
- ROUTINE_SCHEMA : la base de données dans laquelle est définie la procédure ou la fonction ;
- ROUTINE_NAME : le nom de la routine ;
- ROUTINE_TYPE : le type de la routine, qui peut prendre les deux valeurs suivantes :
  a 'FUNCTION' : fonction (procédure stockée renvoyant une valeur) ;
  b 'PROCEDURE' : procédure stockée ;
- DTD_IDENTIFIER : le type de donnée renvoyé (pour les fonctions) ;
- ROUTINE_DEFINITION : le code du corps de la fonction ou de la procédure stockée (toutes les instructions entre les mots clefs BEGIN et END) ;
- CREATED : date et heure de création ;
- LAST_ALTERED : date et heure de dernière modification ;
- DEFINER : l'utilisateur ayant créé la routine.

IX-B - Triggers

Le catalogue fournit également, depuis la version 5.0.10 de MySQL, des informations sur les déclencheurs (ou **triggers**).
On obtient ces informations par la requête suivante :

```
SELECT *
FROM INFORMATION_SCHEMA.TRIGGERS
```

Parmi les données renvoyées par cette requête, on signalera notamment :

- TRIGGER_SCHEMA : la base de données dans laquelle figure le déclencheur ;
- TRIGGER_NAME : le nom du déclencheur ;
- EVENT_MANIPULATION : l'évènement sur lequel se déclenche le trigger, qui peut être soir INSERT, soit UPDATE (en MySQL 5.0, il n'existe pas encore de trigger sur l'instruction DELETE);
- EVENT_OBJECT_SCHEMA : la base contenant la table sur laquelle porte le déclencheur ;

- EVENT_OBJECT_TABLE : la table sur laquelle porte le déclencheur ;
- ACTION_ORDER : l'ordre de déclenchement du trigger dans la liste des triggers portant sur le même objet. En version 5.0, cette valeur est limitée à 0, puisqu'il ne peut y avoir qu'un seul déclencheur par table ;
- ACTION_STATEMENT : le code exécuté à l'appel du déclencheur ;
- ACTION_ORIENTATION : en version 5.0, seul les triggers au niveau ligne étant disponibles, cette colonne ne contient que la valeur 'ROW'. Cela devrait évoluer avec les futures versions de MySQL, si les triggers "instruction" (qui se déclenchent une seule fois et non pas pour chaque ligne) sont implémentés ;
- ACTION_TIMING : le moment de déclenchement du trigger, qui peut être soit avant (BEFORE) soit après (AFTER) l'évènement déclenchant ;
- DEFINER : l'utilisateur ayant créé le trigger.

X - Informations sur les privilèges dans la base de données

Quatre vues système fournissent des informations sur les privilèges globaux sur un serveur MySQL :

- USER_PRIVILEGES ;
- SCHEMA_PRIVILEGES ;
- TABLE_PRIVILEGES ;
- COLUMN_PRIVILEGES.

La vue système **USER_PRIVILEGES** fournit des informations très générales sur les utilisateurs déclarés.
Ces informations sont issues de la table MYSQL.USER.
Les colonnes de cette vue sont les suivants :

- GRANTEE : le nom d'utilisateur et l'hôte de connexion, séparés par un @. Dans MySQL, les droits sont attribués à un nom d'utilisateur, se connectant depuis un hôte (une adresse IP ou '%' pour autoriser une connexion depuis n'importe quel hôte). C'est cette combinaison d'informations qui permet de définir les droits ;
- PRIVILEGE_TYPE : le type de privilèges accordés à l'utilisateur, parmi lesquels on trouve notamment :
  a 'SELECT' : droit de sélection de données ;
  b 'INSERT' : droit d'insertion de données ;
  c 'UPDATE' : droit de mise à jour de données ;
  d 'DELETE' : droit de suppression de données ;
  e 'CREATE' : droit de création d'objets (bases ou tables) ;
  f 'DROP' : droit de suppression d'objets (bases ou tables) ;
  g 'FILE' : lire ou écrire des fichiers sur le serveur (commandes LOAD DATA... INFILE ou SELECT ... INTO OUTFILE) ;
  h 'INDEX' : droit de créer ou supprimer des index ;
  i 'CREATE TEMPORARY TABLES' : droit de créer des tables temporaires ;
  j 'EXECUTE' : droit d'exécuter des procédures stockées ou des fonctions ;
  k 'CREATE VIEW' : droit de créer des vues ;
  l 'CREATE ROUTINE' : droit de créer des procédures stockées, des fonctions ou des triggers ;
  m 'ALTER ROUTINE' : droit de modifier des procédures stockées, des fonctions ou des triggers ;
  n 'CREATE USER' : droit de créer des utilisateurs ;
  o 'USAGE' : équivalent à « aucun privilège », si ce n'est le droit de se connecter au serveur ;
- IS_GRANTABLE : indique si l'utilisateur peut lui-même octroyer les privilèges dont il dispose. Dans la commande GRANT de MySQL, l'option "WITH GRANT OPTION" donne à l'utilisateur la possibilité d'octroyer le privilège qu'il possède à d'autres utilisateurs. C'est cette option qui est décrite ici. Cette information (sous forme de chaîne de caractères) peut prendre 2 valeurs :
  a 'NO' : l'utilisateur ne peut pas octroyer ses privilèges ;
  b 'YES' : l'utilisateur peut octroyer ses privilèges.

La vue système **SCHEMA_PRIVILEGES** donne des informations sur les droits des utilisateurs au niveau de chaque base de données présente sur le serveur.

On y trouve les champs suivants :

- GRANTEE : Le nom de l'utilisateur et son hôte de connexion ;
- TABLE_SCHEMA : le nom de la base de données ;
- PRIVILEGE_TYPE : le type de privilège (voir liste précédente) ;
- IS_GRANTABLE : indique si l'utilisateur peut octroyer ses privilèges.

La vue système **TABLE_PRIVILEGES** donne des informations sur les droits des utilisateurs au niveau de chaque table d'une base de données.
On y trouve les champs suivants :

- GRANTEE : Le nom de l'utilisateur et son hôte de connexion ;
- TABLE_SCHEMA : le nom de la base de données ;
- TABLE_NAME : le nom de la table ;
- PRIVILEGE_TYPE : le type de privilège de table (SELECT, INSERT, UPDATE, ALTER, DROP, INDEX et CREATE VIEW) ;
- IS_GRANTABLE : indique si l'utilisateur peut octroyer ses privilèges.

La vue système **COLUMN_PRIVILEGES** donne des informations sur les droits des utilisateurs au niveau de chaque colonne.
On y trouve les champs suivants :

- GRANTEE : Le nom de l'utilisateur et son hôte de connexion ;
- TABLE_SCHEMA : le nom de la base de données ;
- TABLE_NAME : le nom de la table ;
- COLUMN_NAME : le nom de la colonne ;
- PRIVILEGE_TYPE : le type de privilège de colonne (SELECT, INSERT, et UPDATE) ;
- IS_GRANTABLE : indique si l'utilisateur peut octroyer ses privilèges.

XI - Jeux de caractères et collations

La dernière partie du dictionnaire décrite ici, et non des moindres, est un ensemble de trois vues système portant sur les jeux de caractères et collations.
La première vue, **CHARACTER_SETS**, donne la liste des jeux de caractères disponibles sur le serveur.
Les colonnes de cette vue sont les suivantes :

- CHARACTER_SET_NAME : le nom du jeu de caractères (par exemple ASCII, LATIN2, UTF8...) ;
- DEFAULT_COLLATE_NAME : la collation par défaut associée au jeu de caractère ;
- DESCRIPTION : une description (sommaire) du jeu de caractères ;
- MAXLEN : le nombre d'octets sur lequel est codé le jeu de caractères (par exemple, 3 octets pour l'UTF8).

La deuxième vue système, **COLLATIONS**, fournit des informations sur les collations (ou interclassements) de chaque jeu de caractères.
On y trouve les colonnes suivantes :

- COLLATION_NAME : le nom de la collation ;
- CHARACTER_SET_NAME : le nom du jeu de caractères auquel se rapporte la collation ;
- ID : identifiant numérique de la combinaison jeu de caractères /collation ;

- IS_DEFAULT : indique si la collation est celle par défaut du jeu de caractères auquel elle se rapporte. Cette information (sous forme de chaîne de caractères) peut prendre 2 valeurs :
  - 'YES' : la collation est celle par défaut ;
  - NULL (signifie NON) : la collation n'est pas la collation par défaut ;
- IS_COMPILED : indique si la collation est compilée dans le serveur MySQL. Cette information (sous forme de chaîne de caractères) peut prendre 2 valeurs :
  - 'YES' : la collation est compilée par défaut ;
  - NULL (signifie NON) : la collation n'est pas compilée ;
- SORTLEN : donne une valeur numérique proportionnelle au taux de mémoire requis pour trier une chaîne de caractère du jeu de caractères correspondant selon la collation.

La troisième vue système, **COLLATION_ CHARACTER_ SET_ APPLICABILITY** liste les jeux de caractères applicables pour chaque collation.
Ses données sont les mêmes que les deux premières colonnes de la vue système précédente (COLLATION_NAME et CHARACTER_SET_NAME).
Remarque : pour plus d'informations sur les jeux de caractères et collations, n'hésitez pas à vous reporter au tutoriel suivant : **Les jeux de caractères et collations**

Conclusion

Le catalogue, ou dictionnaire de données, de MySQL offre déjà un grand nombre de possibilités.
Il est d'ailleurs à la base des métadonnées fournies aux langages de programmations interrogeant MySQL par différentes API.
Nul doute que les prochaines versions de MySQL sauront encore l'améliorer.

Remerciements

Merci à **Antoun**, **Cybher**, **Kazou** et **Yiannis** pour leurs relectures et corrections de ce tutoriel, ainsi que leurs précieux conseils.