

Jeux de caractères et collations sous MySQL 5

Cet article est extrait du **Guide complet MySQL 5**, (éditions MicroApplication) et adapté par l'auteur pour Developpez.com, avec l'autorisation de l'éditeur.

Peut-être croyez-vous que les données textuelles sont simples, et qu'il n'y a pas grand-chose à faire avec elles ? Détrompez-vous ! Dès que l'on rentre dans ses subtilités, le texte est une donnée au moins aussi complexe que le temps, et il appelle des traitements spéciaux¹.



- De quelles subtilités s'agit-il ? Pour les illustrer, supposons que vous vous appeliez Leïla :
- si un ami vous cherche dans un annuaire sous l'orthographe « Leila », vous trouvera-t-il ?
- pourquoi votre prénom se transforme-t-il parfois en Le=EFla ? ou en leÃ la ?
- pouvez-vous inclure dans un texte en caractères latins l'écriture arabe de votre prénom ?

Ces trois questions renvoient au système des *jeux de caractères* et de leurs *collations (interclassements)*, dont cet article vous présente l'implémentation sous MySQL 5.

Je vous présente trois approches du sujet :

- un *tutoriel* vous expose en détails le système
- le *mémento* vous donne les tableaux récapitulatifs des fonctions MySQL mises en œuvre
- l'*atelier* vous propose une série d'exercices de mise en pratique et d'approfondissement.

Sommaire

Tutoriel	2
Notion de jeu de caractères	2
Notion de collation	3
Jeu de caractères et la collation des données	4
Effet de la collation sur les requêtes	7
Jeu de caractères et la collation des requêtes	10
Conflits de jeux et de collations	12
Mémento	15
Les fonctions liées au système des jeux de caractères et collations	15
Atelier : explorer les outils textuels	17
Exercices	17
Solutions commentées	18

1. Le plus connu de ces traitements spéciaux est aussi le plus simple, c'est l'opérateur LIKE et ses jokers. Mais MySQL en propose bien d'autres beaucoup plus puissants : REGEXP (recherche par « expression régulière »), MATCH (recherche « full-text »), SOUNDS LIKE (recherche par ressemblance phonétique).

Tutoriel

Notion de jeu de caractères

Un jeu de caractères (*character set*) est un ensemble de lettres, signes de ponctuations et autres symboles, auxquels est associé un numéro de code. Ainsi, le jeu standard ASCII (*American Standard Code for Information Interchange*) donne-t-il le numéro 65 au A majuscule, le 44 à la virgule et le 13 au retour chariot. C'est un jeu créé pour la langue anglaise, et qui ne comporte que 128 caractères. Il ignore donc les caractères accentués, l'œ, le signe €, sans même parler des caractères grecs, arabes ou japonais².

Le principe « un octet = un caractère » a longtemps prévalu. Il a donné naissance à de nombreux jeux taillés à 256 caractères (certains reprenant l'ASCII pour les 128 premières positions et complétant les 128 suivantes). La série des jeux de caractères normalisés ISO 8859 fonctionne de cette manière. L'ISO 8859-1, également appelé *latin-1* ou *occidental (western)* comprend ainsi (presque) tous les caractères nécessaires pour écrire une vingtaine de langues d'Europe occidentale comme le français et l'anglais, tandis que l'ISO 8859-6 fournit les caractères arabes et que l'ISO 8859-15 (*latin-9*) est une version réactualisée du *latin-1*.

Partant d'une autre philosophie, le projet Unicode a pour but de fournir un jeu de caractères unique pour représenter toutes les langues. Cela permet notamment de gérer des textes multi-alphabétiques, par exemple ceux d'un dictionnaire français-arabe. Comme cela nécessite beaucoup plus de 256 caractères, les jeux Unicode utilisent plusieurs octets. Par souci de compatibilité avec les jeux mono-octet, l'Unicode permet d'utiliser des caractères de taille variable, de 1 à 4 octets pour le jeu UTF-8, potentiellement plus de 4 milliards de caractères différents.

Si, à cause de mauvais paramétrages, un caractère Unicode codé sur plusieurs octets est interprété comme mono-octet, il sera représenté par plusieurs caractères, d'où par exemple la transformation du é en Ã©.

Texte vu par l'ordinateur		Interprété selon le jeu de caractères latin1	Interprété selon le jeu de caractères utf8
En décimal	En hexadécimal		
120 121 122	78 79 7A	xyz	xyz
233 116 233	E9 74 E9	été	chaîne invalide
195 169 116 195 169	C3 A9 74 C3 A9	Ã©tÃ©	été

Le texte représenté sous forme numérique et son interprétation selon le jeu de caractères

MySQL vous permet d'utiliser différents jeux de caractères, mono-octet et multi-octets. Voici quelques commandes pour interroger la liste des jeux installés sur votre serveur MySQL :

```
SHOW CHARACTER SET ;
```

Liste de tous les jeux de caractères disponibles sur votre MySQL

² Notez au passage que le jeu de caractères n'est pas la police de caractères (*character font*) : celle-ci associe à chaque caractère un « glyphe », c'est-à-dire une représentation graphique. Ainsi, les lettres A et A sont-elles deux glyphes du même caractère ASCII 65.

```
SHOW CHARACTER SET LIKE 'latin%' ;
```

Liste des jeux de caractères fondés sur l'alphabet latin

```
SELECT *  
FROM Information_Schema.Character_Sets  
WHERE MaxLen > 1;
```

Liste des jeux de caractères multi-octets

Au cours de ce tutoriel, vous utiliserez les deux jeux de caractères suivants (selon leur nom dans MySQL) :

- le `latin1`, mono-octet, qui est une version de l'ISO 8859-1 légèrement modifiée par MySQL AB (afin d'améliorer la compatibilité avec Windows et d'y ajouter, notamment, le signe € et l'œ) ;
- l'`utf8`, qui est une version limitée à 3 octets du standard UTF-8.

Notion de collation

Commençons par un petit exercice : sauriez-vous trier par ordre alphabétique les mots suivants ?

Alphabet	canons	œuvre
Alphabétique	cañon	œuvré
ALPHABET	canonique	ouvré
Alphabets	canon	oeuvre

Question subsidiaire : si vous demandiez à MySQL de rechercher le mot **oeuvre** dans cette liste, qu'aimeriez-vous qu'il trouve ?

Comme vous pouvez le constater, la réponse à ces deux questions dépend de trois choix de départ :

- le problème classique de la sensibilité à la casse : les majuscules doivent-elles précéder les minuscules, ou bien faut-il considérer *A* et *a* comme de même valeur ?
- La sensibilité aux accents : comptent-ils dans le tri ? Font-ils une différence lors de la recherche ?
- La possibilité qu'un caractère (œ) puisse correspondre à plusieurs (oe) : c'est ce qu'Unicode appelle « l'expansion ».

Notez qu'il est possible de faire des choix différents à l'intérieur d'un seul jeu de caractères (tous les mots ci-dessus peuvent être écrits avec `latin1`). C'est pour formaliser ces choix qu'ont été inventées les *collations* (le terme anglais signifie à la fois « rassemblement » et « comparaison » ; dans le sens informatique présenté ici, *collation* est parfois traduit par *interclassement*). Une collation est liée à un jeu de caractères. Elle donne à la fois l'ordre dans lequel classer les caractères, et si certains d'entre eux doivent être considérés comme équivalents. Chaque jeu de caractère possède plusieurs collations, dont une par défaut.

```
SHOW COLLATION LIKE 'latin1%' ;
```

Liste des collations du jeu latin1

```
SHOW COLLATION LIKE 'utf8%' ;
```

Liste des collations du jeu utf8

Toutes les collations ont un nom qui commence par le jeu de caractère auquel elles sont liées, et se termine par l'une de ces trois abréviations :

- `_bin` comme *binary* : les caractères sont dans l'ordre de leurs numéros de code (ce qui donne d'abord toutes les majuscules, puis toutes les minuscules, puis les lettres accentuées en vrac).
- `_cs` comme *case sensitive* : les caractères sont triés selon le ou les langages de référence, mais de manière sensible à la casse.
- `_ci` comme *case insensitive* : *idem*, mais en ignorant la casse.

Pour ce qui concerne le français, voici les collations que vous pouvez utiliser :

Jeu	Collation	Sensible...		Expansi on	Exemple d'ordre	Com- mentaire
		à la casse	aux ac- cents			
latin1	latin1_bin	Oui	Oui	Non	ABoyZaboyzœàÿ	
	latin1_general_cs				AaàBbOoYyÿZzœ	
	latin1_general_ci	Non	Non		AaàBbOoYyZzœÿ	Collation par défaut de latin1
	latin1_swedish_ci					
	latin1_german1_ci					
utf8	utf8_bin	Oui	Oui	ABoyZaboyzàÿœ		
	utf8_general_ci	Non	Non	AaàBbOoYyÿZzœ	Collation par défaut d'utf8	
	utf8_unicode_ci			Oui	AaàBbOoœYyÿZz	Conforme au standard Unicode

Les collations latin1 à connaître

Sans doute vous étonnez-vous de voir `latin1_swedish_ci` dans cette liste ? Il est vrai qu'elle n'est pas d'une grande pertinence pour le français... Pourtant, comme c'est la collation par défaut de `latin1`, c'est peut-être celle que vous utilisez actuellement ! En effet, MySQL AB est une société suédoise, et la documentation précise benoîtement que `latin1_swedish_ci` est "*probablement utilisée par la majorité des clients de MySQL*".

Quant à `latin1_german1_ci`, elle présente l'avantage d'être insensible aux accents. Après une série de tests, elle semble être la collation `latin1` la plus proche des usages français. Vous trouverez en atelier le processus de test.

Jeu de caractères et la collation des données

Il est logique que vous puissiez faire des choix de jeu de caractères et de collation différents selon les données. Par exemple, le catalogue d'une bibliothèque peut être amené à référencer des ouvrages dans n'importe quelle langue. Il serait donc pratique d'y utiliser `utf8`. À l'inverse, une base de suivi d'audience de sites web va stocker des URL et des informations numériques, le `latin1` y sera amplement suffisant et plus économique.

Ces choix peuvent descendre jusqu'à la colonne. Ainsi, il est plutôt souhaitable qu'une colonne de prénoms soit insensible à la casse et aux accents, afin de faciliter les recherches. À l'inverse, on prend généralement soin de rendre les mots de passe sensibles à la casse, pour un maximum de sécurité.

MySQL a donc mis en place un système de choix à quatre niveaux : les données utilisent le jeu de caractères et la collation de leur colonne. Si le jeu et la collation n'ont pas été spécifiés pour la colonne, ce sont ceux de la table. Si ceux de la table n'ont pas été spécifiés, ce sont ceux de la base, et si ceux de la base n'ont pas non plus été spécifiés, ce sont le jeu et la collation par défaut du serveur. Voilà comment vous vous retrouvez à être partout en `latin1_swedish_ci` !

Serveur (choix par défaut pour les nouvelles bases)	Comment faire le choix de départ ?	Le jeu de caractères est choisi lors de l'installation, la collation est sa collation par défaut
	Comment le voir ?	<code>SELECT @@character_set_server, @@collation_server</code>
	Comment le changer ?	Avec MySQL Administrator, ou bien dans la section <i>serveur</i> du fichier <i>my.ini</i> (default-character-set et default-collation)
Base de données (choix par défaut pour les nouvelles tables)	Comment faire le choix de départ ?	<code>CREATE DATABASE Bibli2 CHARSET utf8 COLLATE utf8_unicode_ci ;</code>
	Comment le voir ?	<code>SHOW CREATE DATABASE Bibli2 ; USE Bibli2 ; SELECT @@character_set_database ;</code>
	Comment le changer ?	<code>ALTER DATABASE Bibli2 CHARSET utf8 COLLATE utf8_unicode_ci ;</code>

Table (choix par défaut pour les colonnes)	Comment faire le choix de départ ?	<pre>CREATE TABLE Lecteurs2 (Compte VARCHAR(255), MotDePasse VARCHAR(255)) CHARSET utf8 COLLATE utf8_general_ci ;</pre>
	Comment le voir ?	<pre>SHOW CREATE TABLE Livres2 ;</pre>
	Comment le changer ?	<pre>ALTER TABLE Lecteurs2 CHARSET utf8 COLLATE utf8_general_ci ;</pre>
Colonne (jeu et collation réels des données)	Comment individualiser une colonne à la création de la table ?	<pre>CREATE TABLE Lecteurs2 (Compte VARCHAR(255), MotDePasse VARCHAR(255) CHARSET latin1 COLLATE latin1_bin) CHARSET utf8 COLLATE utf8_general_ci ;</pre>
	Comment faire le choix lors de l'ajout d'une colonne ?	<pre>ALTER TABLE Lecteurs2 ADD COLUMN Nom VARCHAR(255) CHARSET utf8 COLLATE utf8_unicode_ci ;</pre>
	Comment le voir ?	<pre>SHOW FULL COLUMNS FROM Lecteurs2 ; SELECT CHARSET(Nom), COLLATION(Nom) FROM Lecteurs2 ;</pre>
	Comment le changer ?	<pre>ALTER TABLE Lecteurs2 MODIFY MotDePasse VARCHAR(255) CHARSET utf8 COLLATE utf8_bin ;</pre>

Choix du jeu de caractères et de la collation

Pour bien comprendre le système, il faut différencier le niveau colonne des niveaux supérieurs :

- Changer le jeu de caractères et la collation aux niveaux supérieurs (serveur, base ou table) n'a pas de conséquence immédiate : seuls les nouveaux objets créés utiliseront ces valeurs par défaut.
- Changer la collation d'une colonne, en restant dans le même jeu de caractères, se fait sans difficulté. En effet, cela n'affecte pas les données elles-mêmes, mais la façon dont elles sont traitées. Lors du prochain `Select`, vous verrez que les tris et recherches se comportent différemment.
- Changer le jeu de caractères d'une colonne convertit les données vers le nouveau jeu. Cela peut poser un problème si certains caractères de l'ancien jeu n'ont pas d'équivalent dans le nouveau jeu (par exemple, si vous passez une colonne `utf8` en `latin1`).

Même si c'est techniquement possible, il est préférable de ne pas mélanger des jeux de caractères différents au sein d'une même base. Cela permet ainsi de faciliter les comparaisons entre toutes les données de la base.

Enfin, notez qu'avec un jeu de caractères multi-octets, le type `Varchar` prend moins de place que le type `Char`, contrairement à ce qui se passe en mono-octet.

Conversion entre jeux de caractères

Interpréter un texte selon un jeu de caractères, c'est lire les valeurs numériques correspondant aux caractères et les traduire par ces caractères. Ainsi 195 169 est-il interprété comme é en `utf8` et comme Å© en `latin1`.

Au contraire, convertir un texte, c'est effectuer un transcodage. Cela suppose de connaître le jeu de départ et ses équivalences avec le jeu d'arrivée. Ainsi, convertir un 195 169 de l'`utf8` vers le `latin1` donnera 233. Dans les deux cas, c'est bien le caractère é (plus formellement décrit comme « *lettre minuscule latine E accent aigu* ») représenté par deux codes numériques différents.

Les conversions posent problème lorsqu'un caractère n'a pas d'équivalent dans le jeu d'arrivée, ou parfois lorsque MySQL ne connaît pas son équivalent. Ainsi, l'œ `utf8` ne se convertit pas correctement en `latin1`.

Une colonne `Char`, `Varchar` ou `Text` est automatiquement convertie lorsqu'on change son jeu de caractères. MySQL a prévu une instruction permettant de modifier le jeu de caractères (et la collation) d'une table et de toutes ses colonnes `Char`, `Varchar` et `Text`, en convertissant les données. Supposons par exemple que vous disposiez d'une table nommée *Livres* qui soit en `latin1`, et que vous souhaitiez la passer en Unicode :

```
ALTER TABLE Livres
  CONVERT TO CHARSET utf8 COLLATE utf8_unicode_ci ;
```

Conversion de la table Livres en utf8 (avec la collation utf8_unicode_ci)

Il est également possible de demander une conversion dans une requête, avec la fonction `Convert(... Using)` :

```
CREATE TABLE Villes_Unicode
SELECT DISTINCT CONVERT(Ville USING utf8)
  COLLATE utf8_unicode_ci AS Ville
FROM Lecteurs ;
```

Création d'une liste des villes, convertie en utf8 (avec la collation utf8_unicode_ci)

Effet de la collation sur les requêtes

Premiers tests en latin1

Afin d'illustrer l'effet des collations, nous allons utiliser la table *Lecteurs* contenant le fichier des lecteurs d'une bibliothèque, avec leurs adresses. Vous pouvez télécharger le code de cette table ici :

<http://antoun.developpez.com/mysql5/jeux-collations/codes.sql>.

Si vous avez l'installation par défaut de MySQL, vous aurez la collation suédoise pour la colonne *Ville*. Nous allons utiliser une collation plus naturelle :

```
ALTER TABLE Lecteurs
  MODIFY Ville VARCHAR(255) COLLATE latin1_general_ci ;
```

Changement de collation en restant dans le même jeu de caractères

Cherchez les lecteurs parisiens :

```
SELECT Nom, Prenom, Ville
FROM Lecteurs
WHERE Ville = 'Paris' ;
```

Nom	Prenom	Ville
Karimova	Olga	Paris
Lemarchand	Julien	Paris
Von Schmuck	Eliette	Paris
Legrand	Isabelle	Paris
Brown	Charlie	PARIS

Recherche par ville, insensiblement à la casse

Vous pouvez forcer la sensibilité à la casse de la recherche, en utilisant l'opérateur Binary. Cet opérateur indique en fait à MySQL qu'il doit utiliser la collation binaire (celle dont le nom se termine par `_bin`) du jeu de caractères de la colonne concernée. Les deux requêtes suivantes sont donc équivalentes :

```
SELECT Nom, Prenom, Ville
FROM Lecteurs
WHERE Ville = 'Paris'
AND BINARY Ville <> 'Paris' ;
```

Recherche des parisiens avec une casse différente de "Paris"

```
SELECT Nom, Prenom, Ville
FROM Lecteurs
WHERE Ville COLLATE latin1_general_ci = 'Paris'
AND Ville COLLATE latin1_bin <> 'Paris' ;
```

idem, en précisant la collation à utiliser pour chaque condition

Nom	Prenom	Ville
Brown	Charlie	PARIS

Différences entre condition sensible et condition insensible à la casse

Comptez maintenant combien vous avez de lecteurs dans chaque ville :

```
SELECT Ville, COUNT(*) AS NbLecteurs
FROM Lecteurs
GROUP BY Ville ;
```

Regroupement insensible à la casse, sensible aux accents, sans expansion

Ville	NbLecteurs
Paris	5
SAINT-MANDE	1
Saint-Mandé	1
Villejuif	1

Les occurrences Paris et PARIS sont bien reconnus comme la même ville, mais vous avez deux Saint-Mandé (problème d'accent).

*Espaces en fin de texte
et autres subtilités de la comparaison de textes*

Une erreur de saisie fréquente est l'ajout involontaire d'un espace à la fin d'un texte (*trailing space*). Comme l'espace est invisible, c'est une erreur indécélable pour l'utilisateur non averti.

Pour parer à ce problème, la norme SQL définit que les espaces en fin de chaîne littérale et de `Varchar` doivent être ignorés dans les comparaisons textuelles. Ainsi, MySQL considérera que 'Paris' et 'Paris ' sont égaux. La plupart du temps, ce comportement est très pratique... à condition de connaître précisément ses limites :

- `=`, `In`, `<>`, `!=`, `<=`, `>=`, `<`, `>` et `<=>` ignorent les espaces finaux
- `Collate` respecte ce comportement, même avec une collation binaire
- par contre, `Binary` prend toujours en compte les espaces finaux, quel que soit l'opérateur. 'Paris' est donc différent de `Binary` 'Paris '
- l'opérateur `Like` travaille caractère par caractère, et prend donc en compte les espaces finaux (du coup, les deux versions de Paris sont jugées égales par `=`, mais pas ressemblantes par `Like`) ; pour la même raison, `Like` ne prend pas en compte les expansions, quelle que soit la collation (donc 'œ' `LIKE` 'oe' est faux même quand 'œ' = 'oe' est vrai)
- l'opérateur `Regexp` (cf note 1) travaille lui aussi selon ses propres règles, et prend en compte les espaces finaux

Le type `Char` est légèrement différent. En effet, il ajoute lui-même des espaces en fin de texte afin de garder un nombre de caractères fixes. Les espaces finaux sont donc toujours considérés comme non significatifs, qu'ils viennent de l'utilisateur ou du système. Le type `Char` ignore donc systématiquement les espaces finaux lors des comparaisons, même avec `Binary`, `Like` ou `Regexp`.

Enfin, le type `Text` prend systématiquement en compte les espaces finaux.

Conversion en utf8

L'utilisation de `latin1_german1_ci` pourrait régler le problème. Toutefois, un employé de la bibliothèque, natif d'Œuf-en-Ternois (Pas-de-Calais) ayant sensibilisé les bibliothécaires à la question de l'expansion du Œ, vous allez devoir passer la colonne en `utf8`. Pour cela, vous avez deux solutions :

```
SELECT Ville, COUNT(*) AS NbLecteurs
FROM Lecteurs
GROUP BY CONVERT(Ville USING utf8) ;
```

Solution 1 : la ville reste en latin1 dans la table, mais on regroupe sur sa conversion en utf8

```
ALTER TABLE Lecteurs
MODIFY Ville VARCHAR(255) CHARSET utf8 ;
SELECT Ville, COUNT(*) AS NbLecteurs
FROM Lecteurs
GROUP BY Ville ;
```

Solution 2 : on convertit une bonne fois pour toutes la colonne Ville ; la requête de regroupement n'a pas besoin d'être modifiée

Ville	NbLecteurs
Paris	5
Saint-Mandé	2

Villejuif	1
-----------	---

Saint-Mandé est maintenant considérée comme une seule ville, qu'elle porte ou non l'accent. La version qui apparaît est la première que MySQL rencontre dans la table.

Quelle que soit la solution choisie, la collation utilisée est la collation par défaut du jeu de caractères utf8, à savoir utf8_general_ci qui est insensible à la casse et aux accents. Vous auriez pu utiliser un Collate pour préciser une autre collation, juste après la fonction Convert() dans la première solution, ou juste après la définition du jeu utf8 dans la seconde.

La collation affecte ainsi non seulement le tri (Order By) et les recherches (Where), mais aussi le regroupement (Group By), le Distinct (et donc l'Union, qui est distincte par défaut), les fonctions d'agrégation comme Max() ou Group_Concat(), etc.

Jeu de caractères et la collation des requêtes

Les variables de session liées aux jeux de caractères et aux collations

Reprenons la requête de recherche des lecteurs parisiens :

```
SELECT Nom, Prenom, Ville
FROM Lecteurs
WHERE Ville = 'Paris' ;
```

Vous savez que les colonnes *Nom* et *Prenom* sont en latin1, avec la collation latin1_swedish_ci, tandis que *Ville* est soit en latin1 avec latin1_general_ci (si vous avez choisi la solution 1 lors du précédent test), soit en utf8 avec utf8_general_ci (si vous avez choisi la solution 2). Quels sont donc le jeu de caractères et la collation de la chaîne littérale 'Paris' ?

```
Select Charset('Paris'), Collation('Paris') ;
```

Affichage du jeu de caractères et de la collation

Charset('Paris')	Collation('Paris')
utf8	utf8_general_ci

Vous obtiendrez des résultats différents selon votre client

Ces valeurs dépendent des paramètres du client. À chaque ouverture de session, MySQL renseigne quatre variables système, selon les indications du client :

- Le jeu de caractères que le client utilise en saisie (ou du moins le jeu qu'il croit utiliser ; voir l'encadré « Problèmes d'accents avec le client texte ») : cette indication est enregistrée dans la variable @@character_set_client.
- Le jeu de caractères utilisé pour la communication entre le client et MySQL (@@character_set_connection)
- la collation par défaut de ce jeu de caractères détermine la @@collation_connection.
- Le jeu de caractères utilisé pour afficher le résultat des requêtes dans le client (@@character_set_results).

Le texte de la requête est interprété selon le jeu du client, puis converti dans le jeu de la connexion. Charset('Paris') et Collation('Paris') indiquent les valeurs reçues par MySQL, donc @@character_set_connection et

@@collation_connection. MySQL envoie ensuite le résultat en utilisant à nouveau le @@character_set_connection, puis en le convertissant ensuite en @@character_set_results.

Normalement, les trois jeux de caractères sont identiques, ce qui évite une partie des problèmes de conversion. Pour connaître la valeur des variables système liées aux jeux et collations, vous pouvez lancer les requêtes suivantes :

```
SHOW VARIABLES LIKE 'char%' ;  
SHOW VARIABLES LIKE 'colla%' ;
```

Affichage de l'ensemble des variables systèmes liées aux jeux de caractères et collations

Outre les quatre variables citées ci-dessus, vous y retrouverez les jeux et collations par défaut du serveur et de la base de données en cours. Quant au @@character_set_system, c'est celui que MySQL utilise pour enregistrer les métadonnées, c'est-à-dire les noms des bases, tables, colonnes, etc. Il s'agit toujours d'utf8.

Modifier les variables de session

Vous êtes bien sûr libre de modifier ces variables, en particulier de choisir une autre collation pour la connexion :

```
SET @@collation_connection = utf8_unicode_ci ;
```

Changement de la collation de la connexion (à l'intérieur du même jeu de caractères)

La commande Set Names permet de modifier d'un coup les trois jeux de caractères du client, de la connexion et des résultats. Elle définit également la collation de la connexion comme la collation par défaut de @@character_set_connection.

```
SET NAMES utf8 ;
```

Passage du client et des résultats en utf8, et de la connexion en utf8 avec utf8_general_ci

Vous ne devez changer les jeux de caractères que si vous pensez qu'ils sont faux ou sujets à variation (notamment avec les API). En effet, indiquer à MySQL un jeu différent de celui que vous utilisez vraiment provoquera une mauvaise interprétation de certains caractères.

Problèmes d'accents avec le client texte sous Windows

Par défaut, le client texte est configuré comme utilisant latin1. Or le client texte est une application DOS, qui, avec certaines versions de Windows, utilise un jeu de caractères DOS, par exemple le DOS West European (cp850) des versions occidentales de Windows.

```
mysql> SELECT Prenom, Uille FROM Lecteurs WHERE IDlecteur IN (6, 7) ;  
+-----+-----+  
| Prenom | Uille |  
+-----+-----+  
| Melody | Saint-Mandú |  
| Lúttítia | SAINT-MANDE |  
+-----+-----+  
2 rows in set (0.02 sec)
```

Erreur d'affichage des accents sous Windows

Le fait que le client texte se déclare, à tort, comme latin1 auprès de MySQL entraîne une erreur d'interprétation des caractères accentués ou spéciaux (ceux qui vous viennent de la base mais aussi ceux que vous frappez). En corrigeant

les variables système afin qu'elles interprètent correctement le texte envoyé par MySQL, vous devriez corriger le problème :

```
SET NAMES cp850 ;
```

```
SELECT Prenom, Ville FROM Lecteurs WHERE Idlecteur IN (6, 7) ;
```

Correction de @@character_set_client, @@character_set_connexion et @@character_results, puis test de la même requête.

Devez-vous faire un Set Names à chaque nouvelle session ? En théorie, il suffirait de modifier l'option default-character-set de la section CLIENT du fichier *my.ini*. Toutefois, le cp850 n'étant pas un jeu compilé, tenter de l'utiliser comme option de départ empêche le client de démarrer...

Il ne vous reste donc plus qu'à trouver une distribution de MySQL avec cp850 compilé. Le caractère compilé ou non d'un jeu de caractère se voit dans un Show Collation.

Les introducteurs

Une autre possibilité, surtout utilisée à titre de test ou pour résoudre certains cas difficiles, est d'indiquer à MySQL quel est le jeu de caractères d'une chaîne littérale, en la faisant précéder d'un *introducteur* :

```
SELECT _latin1'été', _utf8'été', _cp850'été' ;
```

Test du jeu de caractères au moyen des introducteurs

	_latin1'été'	_utf8'été'	_cp850'été'
Si vous êtes en latin1	été	Ã©tÃ©	,t
Si vous êtes en utf8	Ã©tÃ©	été	+®t+®
Si vous êtes en cp850	?t?	?t?	été

Votre vrai jeu de caractères est celui qui représente les accents correctement

Une chaîne précédée d'un introducteur est envoyée telle quelle à MySQL, quelles que soient les variables @@character_set_client et @@character_set_connexion. MySQL l'interprète ensuite selon le jeu indiqué par l'introducteur, puis le renvoie.

Conflits de jeux et de collations

Considérez la requête suivante :

```
SELECT Nom, Prenom, Ville
FROM Lecteurs
WHERE Ville = 'Saint-Mandé' ;
```

La colonne *Ville* a un jeu de caractère et une collation, la chaîne littérale 'Saint-Mandé' également :

```
SELECT CHARSET(Ville), COLLATION(Ville),
CHARSET('Saint-Mandé'), COLLATION('Saint-Mandé')
FROM Lecteurs
LIMIT 1 ;
```

CHARSET(Ville)	COLLATION(Ville)	CHARSET('Saint-Mandé')	COLLATION('Saint-Mandé')
latin1	latin1_general_ci	utf8	utf8_general_ci

Avec, par exemple, Ville laissée en latin1 et un client en utf8

Comparer *Ville* et 'Saint-Mandé' alors qu'ils ont des jeux de caractères différents ne pose pas trop de problème, il suffit de convertir l'un dans le jeu de

l'autre. Mais quelle collation choisir ? Si MySQL choisit la collation de la colonne *Ville*, sensible aux accents, il ne trouvera que Melody. S'il choisit celle de la chaîne littérale 'Saint-Mandé', il trouvera Melody et Lætitia.

Opter pour la collation de l'un des deux termes, c'est forcer (*coerce*) l'autre terme à abandonner sa propre collation. La facilité avec laquelle on peut forcer un terme est nommée « coercibilité ». La collation utilisée sera donc celle du terme le moins coercible :

```
SELECT COERCIBILITY(Ville), COERCIBILITY ('Saint-Mandé')
FROM Lecteurs
LIMIT 1 ;
```

Coercibilités d'une colonne et d'une chaîne littérale

COERCIBILITY(Ville)	COERCIBILITY('Saint-Mandé')
2	4

MySQL choisira la collation de la colonne *Ville*, et ne trouvera que Melody

Voici les différents niveaux de coercibilité prévus par la norme SQL :

- 0 (dit incoercible ou *Explicit*) : élément dont la collation est précisée explicitement par *Collate*.
- 1 : concaténation de deux chaînes de collations différentes.
- 2 (*Implicit*) : colonne.
- 3 : information provenant du système, par exemple résultat de *Database()* ou *User()*.
- 4 (*Coercible*) : chaîne littérale, calcul, résultat d'une fonction.
- 5 : Null.

En cas de conflit entre deux éléments de même coercibilité, la norme indique qu'une erreur doit se produire. Dans la pratique, le comportement de MySQL ne répond pas toujours exactement à la norme, et peut changer d'une sous-version à l'autre. Soyez donc vigilant !

Dans tous les cas le problème se résoud par un *Collate*... qu'il faut placer judicieusement. En effet, les deux conditions ci-dessous sont correctes (dans l'hypothèse choisie où *Ville* est en *latin1* et le client en *utf8*).

```
... Ville = 'Saint-Mandé' COLLATE utf8_unicode_ci
... Ville COLLATE latin1_german1_ci = 'Saint-Mandé'
```

Par contre, ces deux conditions-là tentent d'appliquer une collation qui ne correspond pas au jeu de caractères :

```
... Ville COLLATE utf8_unicode_ci = 'Saint-Mandé'
```

COLLATE est mal placé

```
ERROR 1253 (42000): COLLATION 'utf8_unicode_ci' is not valid
for CHARACTER SET 'latin1'
```

Réponse de MySQL

```
... Ville = 'Saint-Mandé' COLLATE latin1_german1_ci
```

COLLATE est également mal placé

```
ERROR 1253 (42000): COLLATION 'latin1_german1_ci'  
is not valid for CHARACTER SET 'utf8'
```

Réponse de MySQL

Mémento

Les fonctions liées au système des jeux de caractères et collations

Les fonctions d'information

`Charset()` donne le jeu de caractères, `Collation()` la collation et `Coercibility()` la coercibilité.

Les fonctions de codage de caractères

Ces fonctions interprètent les nombres en texte :

Fonction	Description, exemple et résultat	
<code>CHAR(Nb1, Nb2... USING Jeu)</code>	Prend une série de nombres et renvoie la chaîne de caractères correspondante, selon le jeu indiqué (les nombres supérieurs à 255 sont convertis en une série d'octets)	
	<code>CHAR(65, 66 USING latin1)</code>	AB
	<code>CHAR(16706 USING latin1)</code>	AB
	<code>CHAR(65 * 256 + 66 USING latin1)</code>	AB
<code>UNHEX(NbHexa)</code> Correspond aux écritures littérales : <code>0xNbHexa</code> <code>x'NbHexa'</code>	Prend une chaîne de nombres hexadécimaux et l'interprète selon le pseudo-jeu <code>binary</code> (équivalent de <code>latin1</code>)	
	<code>UNHEX('78797A')</code>	xyz
	<code>0x78797A</code>	xyz
	<code>x'4142'</code>	AB

Codage des caractères

Les fonctions de décodage

Ces trois fonctions permettent de connaître le code numérique d'un caractère ou d'un texte :

- `Ascii(Texte)` donne le code numérique du premier caractère du texte. Avec les jeux multi-octets, ne donne que le premier octet.
- `Ord(Texte)` donne le code numérique du premier caractère du texte, y compris avec les jeux multi-octets.
- `Hex(Texte)` transforme le texte entier en une série d'octets (exprimés sous forme de deux chiffres hexadécimaux). C'est la réciproque de `Unhex()`.

Texte	Ascii(Texte)	Ord(Texte)	Hex(Texte)
'AB'	65	65	4142
'é' en utf8	195	50089	C3A9
'Ã' en latin1	195	195	C3

Comparaison des fonctions de décodage

La taille des textes

Fonction	Description, exemple et résultat	
Char_Length(<i>Texte</i>)	Donne le nombre de caractères d'un texte	
	Char_Length(_utf8'été')	3
Length(<i>Texte</i>) Synonyme :	Donne le nombre d'octets d'un texte	
	Length(_utf8'été')	5
Octet_Length(<i>Texte</i>)	Octet_Length(_utf8'été')	5

Codage des caractères

La conversion entre les nombres

Bien qu'elles ne concernent pas spécifiquement les textes, ces fonctions et écritures sont utiles pour passer de la base décimale à la base hexadécimale et réciproquement.

Conversion	Exemples	Résultat
De décimal en hexadécimal	Conv(50089, 10, 16)	C3A9
	Hex(50089)	C3A9
D'hexadécimal en décimal	Conv('C3A9', 16, 10)	50089
	0xC3A9 + 0	50089
	x'C3A9' + 0	50089

Conversions numériques

Atelier : explorer les outils textuels

Exercices

Ces différents exercices sont prévus pour un client en UTF-8 afin de pouvoir représenter l'ensemble des caractères. Les tests ont été effectués avec phpMyAdmin sous Internet Explorer.

Explorer les jeux de caractères et leurs collations

Afin de savoir quels sont les caractères contenus dans chaque jeu, et comment les collations les ordonnent, vous allez utiliser une table nommée *Octet*, qui contient les nombres de 0 à 255. Pour la créer, je vous propose d'utiliser une table *Chiffres* contenant les chiffres de 0 à 9, et des produits cartésiens³ :

```
CREATE TABLE chiffres (Num TINYINT UNSIGNED NOT NULL PRIMARY KEY);

INSERT
  INTO chiffres(num)
  VALUES (0), (1), (2), (3), (4), (5), (6), (7), (8), (9);

CREATE TABLE octet (Num TINYINT UNSIGNED NOT NULL PRIMARY KEY);

INSERT
  INTO octet(Num)
  SELECT c.Num * 100 + d.Num * 10 + u.Num
  FROM chiffres AS c, chiffres AS d, chiffres AS u
  WHERE c.num * 100 + d.num * 10 + u.num <= 255
  ORDER BY 1;
```

Les caractères 0 à 31 sont des caractères de contrôle non représentables, comme le retour à la ligne, la tabulation, etc.

- 1 Rédigez une requête présentant les caractères 32 à 255 du jeu *latin1* avec leur numéro de code, selon l'ordre binaire (c'est-à-dire l'ordre de ces numéros)
- 2 Modifiez cette requête afin qu'elle suive l'ordre de la collation *latin1_general_ci*
- 3 Comme *latin1_general_ci* est insensible à la casse, cette requête présente les caractères équivalents comme *A* et *a* dans un ordre ou dans l'autre; modifiez-la afin que, tout en respectant l'ordre de la collation, les caractères équivalents soient présentés avec la majuscule d'abord
- 4 Vérifiez le jeu de caractères et la collation par défaut de la table *Octet* ; au besoin, modifiez-la afin qu'elle utilise par défaut *latin1_bin*
- 5 Ajoutez deux colonnes *Xnum* et *CarLat1* à la table *Octet*, et placez-y respectivement le code hexadécimal correspondant au *Num*, et le caractère *latin1* correspondant au *Num*
- 6 Vérifiez par une requête que le code hexadécimal de *CarLat1* correspond bien à *Xnum*

Cette table pourra à l'avenir vous servir de référence sur le jeu *latin1*. Vous allez vous en servir pour explorer les différentes collations.

Il est facile de trier *Octet* selon l'une ou l'autre des collations de *latin1*, mais cela ne vous permet pas de voir clairement les équivalences entre les caractères.

³ Ce code est disponible dans le fichier téléchargeable <http://antoun.developpez.com/mysql5/jeux-collations/codes.sql>.

- 7 Rédigez une requête qui présente les équivalences de chacune des lettres majuscules A à Z selon la collation `latin1_general_ci`.
- 8 La requête précédente donne les équivalences, mais son résultat n'est pas très facile à lire. Modifiez-la afin que, pour chacune des 26 lettres de référence, elle présente l'ensemble des équivalents sur une seule ligne.
- 9 Étendez cette requête aux caractères délicats Ç, ç, Ÿ, ÿ, Œ, œ, Æ, æ, ñ, Ñ et ß, et testez les différentes collations de `latin1`.
- 10 Rédigez une nouvelle requête utilisant `Octet` afin de détecter les expansions (un caractère spécial = deux caractères simples) de `latin1_german2_ci` (seule collation de `latin1` avec des expansions).

Vous avez maintenant les outils pour choisir vos collations mono-octet en toute connaissance de cause !

Solutions commentées

Explorer les jeux de caractères et leurs collations

- 1 Rédigez une requête présentant les caractères 32 à 255 du jeu `latin1` avec leur numéro de code, selon l'ordre binaire (c'est-à-dire l'ordre de ces numéros)

Il suffit d'utiliser la fonction de codage `Char()` :

```
SELECT Num, Char(Num USING latin1)
FROM Octet
WHERE Num >= 32
ORDER BY Num
```

Codage du jeu `latin1`

- 2 Modifiez cette requête afin qu'elle suive l'ordre de la collation `latin1_general_ci`

Il suffit d'ajouter un `Collate` :

```
SELECT Num, Char(Num USING latin1)
FROM Octet
WHERE Num >= 32
ORDER BY Char(Num USING latin1) COLLATE latin1_general_ci
```

Ordre de la collation `latin1_general_ci`

Notez que `Order By 2 Collate latin1_general_ci` ne fonctionne pas.

- 3 Modifiez cette requête afin que, tout en respectant l'ordre de la collation, les caractères équivalents soient présentés avec la majuscule d'abord

Il suffit d'ajouter une seconde clé de tri :

```
SELECT Num, Char(Num USING latin1)
FROM Octet
WHERE Num >= 32
ORDER BY Char(Num USING latin1)
        COLLATE latin1_general_ci, Num
```

Idem, avec priorité aux majuscules

Plutôt que `Num`, vous pouvez également utiliser `Char(Num USING latin1) Collate latin1_general_cs` ou `Char(Num USING latin1) COLLATE latin1_bin`.

- 4 Vérifiez le jeu de caractères et la collation par défaut de la table `Octet` ; au besoin, modifiez-la afin qu'elle utilise par défaut `latin1_bin`

Pour connaître le jeu et la collation, vous devez regarder la requête de création de la table :

```
SHOW CREATE TABLE Octet ;
```

Table	Create Table
Octet	CREATE TABLE 'octet' ('Num' tinyint(3) unsigned NOT NULL) ENGINE=InnoDB DEFAULT CHARSET=latin1

Résultat

Certains clients ne permettent pas de visualiser facilement les textes multilignes. Le Show Create sera alors plus lisible dans le client texte, avec \G à la place du point-virgule :

```
SHOW CREATE TABLE Octet \G
```

Requête avec demande d'affichage en lignes (\G doit être en majuscule)

```
***** 1. row *****  
Table: Octet  
Create Table: CREATE TABLE 'octet' (  
  'Num' tinyint(3) unsigned NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1  
1 row in set (0.00 sec)
```

Réponse du client texte

Comme la collation n'est pas précisée, c'est la collation par défaut du jeu de caractères, donc latin1_swedish_ci. Il ne reste qu'à la modifier :

```
ALTER TABLE Octet COLLATE latin1_bin ;
```

Modification de la collation par défaut de la table

5 Ajoutez deux colonnes, Xnum et CarLat1, à la table Octet, et placez-y respectivement le code hexadécimal correspondant au Num, et le caractère latin1 correspondant au Num

XNum est un code qui doit toujours être sur deux caractères. CarLat1 est un caractère unique mono-octet. Il pourrait sembler logique de le mettre en Char(1), mais cela supposerait que l'espace ne compte pas⁴. Afin de garder un comportement cohérent pour tous les caractères, le Varchar(1) s'impose.

```
ALTER TABLE Octet  
ADD Xnum CHAR(2), ADD CarLat1 VARCHAR(1) ;  
UPDATE Octet SET Xnum = HEX(Num),  
  CarLat1 = CHAR(Num USING latin1) ;
```

En regardant la table, vous constaterez que les nombres 0 à 15 ont été convertis sous la forme 0 à F, et non 00 à 0F comme le voudrait la représentation hexadécimale d'un octet. Passez donc la requête de correction suivante :

```
UPDATE Octet  
SET Xnum = CONCAT('0', Xnum)  
WHERE CHAR_LENGTH(Xnum) = 1 ;
```

Ajout d'un zéro au début des 16 premiers codes hexadécimaux

Comme vous êtes en mono-octet, Length() aurait fonctionné de la même manière, ainsi que Like '_' (un caractère quelconque). Enfin, vous auriez aussi pu utiliser plus simplement Num < 16.

⁴ Voir l'encadré *Espaces en fin de texte...*

6 Vérifiez par une requête que le code hexadécimal de *CarLat1* correspond bien à *Xnum*

Vous pouvez par exemple décoder *CarLat1* en hexadécimal et tester si cela permet bien de retrouver *Xnum* :

```
SELECT Num, CarLat1, Xnum, HEX(CarLat1)
FROM Octet
WHERE Xnum = HEX(CarLat1) ;
```

Cette requête semble correcte, pourtant...

```
Illegal mix of collations (latin1_bin,IMPLICIT)
and (utf8_general_ci,COERCIBLE) for operation '='
```

Réponse de MySQL

Le résultat de `Hex()` est produit avec le pseudo-jeu `binary`, que MySQL refuse de comparer avec du `latin1` (ce qui semble plus un bug qu'un comportement conforme à la norme). Vous allez donc lui demander explicitement la conversion :

```
SELECT Num, CarLat1, Xnum, HEX(CarLat1)
FROM Octet
WHERE Xnum = CONVERT(HEX(CarLat1) USING binary);
```

Comparaison entre deux chaînes en binary

```
...
256 rows in set (0.01 sec)
```

Réponse de MySQL : nous retrouvons bien nos 256 caractères

Cette vérification permet de s'assurer qu'aucun artefact n'est intervenu, comme par exemple les 16 premiers codes sans le zéro initial de l'exercice précédent, ou bien la neutralisation de l'espace si vous aviez choisi de mettre *CarLat1* en `Char(1)`.

7 Rédigez une requête qui présente les équivalences de chacune des lettres majuscules A à Z selon la collation `latin1_general_ci`

Il faut pour cela comparer la table *Octet* avec elle-même, en utilisant deux alias de tables. La jointure se fait sur l'égalité entre les colonnes *CarLat1*, selon la collation testée :

```
SELECT L.CarLat1 AS Lettre, E.CarLat1 AS Equivalent
FROM Octet AS L
INNER JOIN Octet AS E
ON L.CarLat1 = E.CarLat1 COLLATE latin1_general_ci
WHERE L.CarLat1 BETWEEN 'A' AND 'Z' -- (latin1_bin implicite)
ORDER BY L.CarLat1 -- (latin1_bin implicite)
```

Équivalences des 26 lettres de A à Z selon `latin1_general_ci`

```
52 rows in set (0.00 sec)
```

A chaque lettre correspond deux caractères équivalents, la lettre majuscule et la lettre minuscule

Remplacez maintenant `Collate latin1_general_ci` par `Collate latin1_swedish_ci` :

```
100 rows in set (0.00 sec)
```

A chaque lettre peut correspondre de nombreuses versions accentuées

Vous pouvez constater que selon la collation suédoise, *ü* est équivalent à *y*. C'est également le cas avec `latin1_danish_ci`.

8 Modifiez la requête précédente afin que, pour chacune des 26 lettres de référence, elle présente l'ensemble des équivalents sur une seule ligne.

Il faut utiliser la fonction d'agrégation `Group_Concat()`⁵, spécifique à MySQL :

```
SELECT L.CarLat1 AS Lettre,
       GROUP_CONCAT(E.CarLat1
                    ORDER BY E.CarLat1 COLLATE latin1_general_ci
                    SEPARATOR '') COLLATE latin1_general_ci -- explication ci-dessous
       AS Equivalents
FROM Octet AS L
     INNER JOIN Octet AS E
     ON L.CarLat1 = E.CarLat1 COLLATE latin1_swedish_ci
WHERE L.CarLat1 BETWEEN 'A' AND 'Z'
GROUP BY L.CarLat1
ORDER BY L.CarLat1
```

Le `COLLATE` placé juste avant `AS Equivalents` a pour seul rôle d'éviter que le résultat du `GROUP_CONCAT()` ne soit interprété comme un Blob (ce qui semble un bug de la version testée).

9 Étendez cette requête aux caractères délicats Ç, ç, Ÿ, ÿ, Æ, æ, Æ, æ, ñ, Ñ et ß, et testez les différentes collations de `latin1`

Afin de parer à toute mauvaise interprétation des caractères, commencez par rechercher les codes correspondant :

```
SELECT CarLat1, Num
FROM Octet
WHERE CarLat1 IN ('A', 'Z', 'Ç', 'ç', UPPER('ÿ'), 'ÿ',
                 'Æ', 'æ', 'Ñ', 'ñ', 'ß') ;
-- UPPER pour Windows, où le Ÿ ne peut se saisir
-- que par le code Alt+0159, (ce qui supposerait que
-- vous connaissiez déjà le code de ce caractère)
```

Recherche des codes latin1 correspondant à A, Z et 11 caractères spéciaux

Il suffit maintenant de modifier la clause `Where` avec les codes obtenus, et de trier la requête selon la collation testée.

```
SELECT L.CarLat1 AS Lettre,
       GROUP_CONCAT(E.CarLat1
                    ORDER BY E.CarLat1
                    SEPARATOR '')
       COLLATE latin1_swedish_ci AS Equivalents
FROM Octet AS L
     INNER JOIN Octet AS E
     ON L.CarLat1 = E.CarLat1 COLLATE latin1_swedish_ci
WHERE L.Num BETWEEN 65 AND 90
     OR L.Num IN (140, 156, 159, 198, 199, 209, 223, 230, 231, 241, 255)
GROUP BY L.CarLat1
ORDER BY L.CarLat1 COLLATE latin1_swedish_ci
```

Requête de test d'une collation

⁵ Pour ceux qui ne la connaissent pas, disons que `Group_Concat()` est à `Concat()` ce que `Sum()` est à `+`. Elle dispose de trois options, `DISTINCT` (comme les autres fonctions d'agrégation), `ORDER BY` et `SEPARATOR`.

Pour tester une autre collation, il suffit bien sûr de remplacer les trois mentions de `latin1_swedish_ci` par la collation à tester. Vous pouvez constater les points suivants :

- Toutes les collations insensibles aux accents placent néanmoins *Æ*, *œ*, *ÿ* et *ÿ* après le *Z*. Elles ne détectent pas non plus l'équivalence entre les versions majuscules et minuscules de ces caractères.
- `latin1_spanish_ci` considère *Ñ* et *ñ* comme différentes de *N* et *n*.
- `latin1_danish_ci` considère (comme `latin1_swedish_ci`) qu'*Ü* et *ü* sont équivalents à *Y* et à *y*.
- `latin1_german1_ci` considère *ß* comme équivalent à *S* et à *s*.

Au final, `latin_german1_ci` et `latin_german2_ci` semblent les mieux placées pour un ordre alphabétique français. La question suivante va permettre de les départager.

10 Rédigez une nouvelle requête utilisant Octet afin de détecter les expansions (un caractère spécial = deux caractères simples) de latin1_german2_ci (seule collation de latin1 avec des expansions).

Il faut cette fois-ci comparer un caractère avec la concaténation de deux autres, soit donc trois alias de la table *Octet*. Le premier alias donnera la lettre testée, vous y exclurez les caractères de contrôle (0 à 31) et l'espace (32). Cette lettre sera comparée avec toutes les combinaisons des caractères simples des deux autres alias d'*Octet*, soit donc un produit cartésien entre les ensembles des caractères 65 à 90 de ces deux tables.

```
SELECT DISTINCT -- permet de confondre maj. et minuscules
  L.CarLat1 COLLATE latin1_german2_ci,
-- COLLATE pour le DISTINCT
  CONCAT(C1.CarLat1, C2.CarLat1)
  COLLATE latin1_german2_ci -- id.
FROM Octet AS L
  INNER JOIN (Octet AS C1 CROSS JOIN Octet AS C2)
  ON L.CarLat1 = CONCAT(C1.CarLat1, C2.CarLat1)
  COLLATE latin1_german2_ci
WHERE L.Num > 32
  AND C1.Num BETWEEN 65 AND 90
  AND C2.Num BETWEEN 65 AND 90
```

Détection des expansions

Vous trouvez donc l'expansion de *Ä* en *AE*, *Ö* en *OE*, *Ü* en *UE* et *ß* en *SS* (mais rien qui vous aiderait à gérer *Æuf-en-Ternois*, qui a effectivement besoin de l'*utf8*).

Le fait d'ôter `Distinct` et de remplacer `Between 65 And 90` par `> 32` permettrait de visualiser les 604 combinaisons de majuscules, minuscules et accents de ces quatre expansions.

Antoine Dinimant

Cet article est extrait du Guide complet MySQL 5, et adapté par l'auteur pour Developpez.com, avec l'autorisation de l'éditeur.

© *MicroApplication 2006*

Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de MICRO APPLICATION est illicite (article L122-4 du code de la propriété intellectuelle).

Cette représentation ou reproduction illicite, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles L335-2 et suivants du code de la propriété intellectuelle.

Le code de la propriété intellectuelle n'autorise aux termes de l'article L122-5 que les reproductions strictement destinées à l'usage privé et non destinées à l'utilisation collective d'une part, et d'autre part, que les analyses et courtes citations dans un but d'exemple et d'illustration.