Datenbank: MySQL 4.1 und Zeichensätze

MySQL 4.1 und Zeichensätze

MySQL 4.1 und Zeichensätze

Bei der Verwendung von MySQL 4.1 und höher, kommt es immer wieder zu Problemen mit der Darstellung von
Web-Seiten in verschiedenen Zeichensätzen. Dies liegt an der wichtigsten Neuerung ab MySQL 4.1

gegenüber den Vorgänger-Versionen, Unicode-fähig
zu sein. Dadurch ist es erheblich einfacher geworden,
mehrsprachige Web-Seiten anzubieten.

Der richtige Zeichensatz in HTML-Dokumenten

Ein weit verbreiteter und oft ausreichender Zeichensatz für z.B. westeuropäischen Inhalt ist ISO-8859-1 (auch
bekannt unter „Latin-1”). Er stammt aus der ISO-8859-Familie und beinhaltet unter Anderem die
deutschen Umlaute und einige relevante Sonderzeichen. Wird das Dokument mit diesem Zeichensatz
ausgeliefert, können alle darin enthaltenen Zeichen, mit Ausnahme einiger Zeichen, die in HTML eine
besondere Bedeutung besitzen (z.B. ‹, › und &), direkt im HTML-Quelltext verwendet werden.

Da ISO-8859-1 nur ein Byte zur Kodierung eines Zeichens verwendet, können damit nur 256 Zeichen
dargestellt werden. Was tut man aber, möchte man auf seiner Web-Seite Zeichen darstellen, die in dem
gewählten Zeichensatz nicht enthalten sind? Eine Möglichkeit ist es, diese Zeichen mit Hilfe einer speziellen numerischen Notation
einzufügen. Mit dieser Notation lässt sich
jedes Zeichen aus dem Unicode-Zeichenvorrat notieren. Das große, griechische Sigma (Σ) z.B. ist

in ISO-8859-1 nicht enthalten, es kann also nicht im Quelltext verwendet werden, der Browser würde
ein anderes Zeichen darstellen. Um es dennoch darzustellen, könnte die angesprochene numerische
Notation verwendet werden, in diesem Fall »Σ«. Diese Praktik ist jedoch, wenn überhaupt, nur für einzelne Zeichen sinnvoll.

UTF-8 für mehrsprachige Web-Seiten

Sollen ganze Sätze einer anderen Sprache dargestellt werden, bietet sich ein Zeichensatz mit einem größeren Zeichenvorrat an. UTF-8 z.B. kodiert
Zeichen nicht in einem, sondern in einer Variablen Länge von Bytes - dadurch lassen sich alle Zeichen des Unicode-Systems darstellen. Sie können also direkt in den HTML-Quelltext notiert werden,
sofern der verwendete Editor selbst UTF-8 unterstützt.

Mit welchem Zeichensatz die empfangene Ressource zu dekodieren ist, erfährt der anfordende Browser vom Webserver. Dazu muss der Webserver erfahren, dass er diese Angabe mitsenden soll, was
auf verschiedene Arten geschehen kann:

- in PHP steht dazu die Funktion header() zur Verfügung. Mittels »header('Content-Type: text/html; charset=utf-8');« wird dem Webserver der verwendete MIME-Typ und Zeichensatz mitgeteilt.

- in einer .htaccess-Datei lässt sich dem Webserver mittels AddCharset mitteilen, bestimmte Ressourcen mit einem definierten Zeichensatz auszuliefern. So veranlasst die Zeile »AddCharset utf-8 .html .php« den Webserver, sämtliche Dokumente mit der Datei-Endung html und php als Ressourcen mit dem Zeichensatz UTF-8 auszuliefern. Möchte man sämtlichen Antworten, die mit dem MIME-Typ text/plain und text/html ausgeliefert werden, außerdem den Zeichensatz UTF-8 vorgeben, genügt eine Zeile, die die Anweisung AddDefaultCharset enthält: »AddDefaultCharset utf-8«

Damit auch nach dem Abspeichern des Dokuments noch der korrekte Zeichensatz verwendet wird, gibt
es außerdem die im HTML-Quelltext zu notierende Meta-Angabe zum Zeichensatz
»http-equiv="content-type"«.
Diese sollte zwar auch vom Webserver interpretiert werden, allerdings nur, wenn kein HTTP-Header angegeben ist - es würde keinen Sinn ergeben, eine hiervon abweichende Angabe zu notieren. Im Konfliktfall sollte vom Browser die Angabe im HTTP-Header den Vorzug erhalten, was dann zu der Merkwürdigkeit führen könnte, dass das gleiche Dokument, über den Webserver ausgeliefert, im Unterschied zu einem lokalen Aufruf anders dekodiert würde, denn lokal aufgerufen steht dem Browser
natürlich kein HTTP-Header zur Verfügung und er würde sich wohl an die Angabe im Meta-Tag halten.
Die Information, ob und welche Zeichensatz-Angabe im HTTP-Header enthalten ist, kann auf mehrere
Arten festgestellt werden: viele Browser zeigen die vom Server mitgelieferte Kodierungsangabe in ihren Seiteninformationen. Außerdem gibt es sogenannte HTTP-Trace-Services wie den Web-Sniffer, welche die
Header-Informationen anzeigen.

Der Unterschied zwischen MySQL =‹4.0 und =›4.1

MySQL 4.0 und frühere Versionen unterstützten lediglich Zeichensätze, deren Zeichen mit je 8 Bit kodiert werden. Der verwendete Zeichensatz wurde auf Server-Ebene festgelegt und entsprach standardmäßig Latin-1. Es wurden weder Information zum tatsächlich verwendeten Zeichensatz und Sortiervorschriften der Daten gespeichert, noch beherrschte MySQL eine Konvertierung der Daten in
andere Zeichensätze. Um dennoch mit UTF-8 zu arbeiten, wurden einfach die UTF-8-Bytesequenzen
gespeichert, also z.B. „MÃ¼ller” statt „Müller”, denn das ü wird in UTF-8
mit zwei Bytes, 188 und 195 kodiert, welche als Latin-1 dekodiert Ã bzw. ¼ entsprechen. Dadurch ergaben sich einige Nachteile: die Daten wurden von MySQL nicht mehr korrekt sortiert, die Volltext-Suche war nicht mehr komplett benutzbar, und vieles mehr.

Seit MySQL 4.1 gibt es eine viel größere Unterstützung für verschiedene Zeichensätze. Die Daten werden intern standardmäßig in UTF-8 kodiert abgelegt, die Daten zwischen Server und Client aber weiterhin in Latin-1 ausgetauscht. Für eine einsprachige Webseite, die ISO-8859-1 nutzt, ändert sich
also erstmal nichts. Soll die Webseite in UTF-8 sein, muss man dem Datenbankserver
mitteilen, dass die Daten fortan

als UTF-8-kodiert ankommen und auch ausgeliefert werden sollen. Das geschieht mit der SQL-Anweisung
»SET NAMES 'utf8'« oder »SET CHARACTER SET utf8«.

Die gleichen Angaben müssen auch beim Im-/Export von Daten, z.B. über den phpMyAdmin, angegeben
werden. Bei DB-Dumps, die von einem MySQL 4.0-Server erstellt wurden, wird der Zeichensatz der Daten
(bei deutschsprachigen Daten i.d.R. Latin-1) unter „Zeichencodierung der Datei” ausgewählt.

„MÃ¼ller” statt „Müller”

Trotzdem kommt es vor, dass auf MySQL 4.1 aufbauende Web-Anwendungen die Umlaute verhunzen. Das
ist in der Regel eine Folge von nicht korrekt kodiert übertragenen bzw. gespeicherten Daten. Haben wir beispielsweise ein HTML-Formular, muss dem Browser mitgeteilt werden, welchen Zeichensatz er für
dieses Formular akzeptieren darf. Soll die Web-Seite als ISO-8859-1 kodiert ausgeliefert werden, der
Browser sendet aber fälschlicherweise UTF-8, kommt es zu oben genannter Abweichung: der Browser
sendet die Byte-Sequenz, die in ISO-8859-1 „MÃ¼ller” entspricht. Gelangt diese Bytesequenz ohne Überprüfung in die Datenbank, setzt sich der Fehler fort. Um das zu verhindern, gibt es mehrere Möglichkeiten:

- der Browser darf nur ISO-8859-1 akzeptieren: im einleitenden Formular-Element benutzt man dazu
  das Attribut accept-charset mit dem
  Wert ISO-8859-1,
  also z.B. »‹?php

  <form action="speichern.php" method="post" accept-charset="ISO-8859-1">

  ?›«.
  Dieses Vorgehen hat den Nachteil, dass einige Browser Fragezeichen für eingegebene Zeichen außerhalb
  von ISO-8859-1 senden. Der Speichervorgang in einem verarbeitenden PHP-Script könnte beispielsweise
  mit folgenden Zeilen geschehen:

```
‹?php

$link = mysql_connect(&#39;host&#39;, &#39;user&#39;, &#39;pass&#39;)

or die (&#39;Verbindung zur DB nicht möglich&#39;);

mysql_query(&#39;SET NAMES \&#39;latin1\&#39;&#39;, $link);

mysql_query(&#39;INSERT INTO datenbank.tabelle (spalte) &#39; .

&#39;VALUES (\&#39;&#39; . mysql_real_escape_string($daten, $link) . &#39;\&#39;)&#39;,
$link)

or die(&#39;INSERT fehlgeschlagen: &#39; . mysql_error($link));

?›
```

- man akzeptiert und speichert generell UTF-8, indem im Formular-Element »accept-charset="UTF-8"« und unter MySQL die Anweisung »SET NAMES 'utf8'« genutzt werden.
  Aber auch hier können bei der Auslieferung mittels Latin-1 Zeichen verloren gehen, da Latin-1 einen
  kleineren Zeichenvorrat als UTF-8 besitzt.

Die Sortierung (collation) der Daten

Die Standardeinstellung für die Sortiermethode auf unseren Servern entspricht „latin1_swedish_ci”. Dies liefert die für die meisten Kunden erwartete Sortiermethode.
Dies gilt bei der Verwendung von UTF-8 nicht, hier gibt es eigene Sortiervorschriften. Bei der Verwendung der standardmäßigen „utf8_general_ci” wird aber augenscheinlich „falsch” sortiert, Umlaute und das ß wandern nach hinten, da hier nach den Byte-Werten sortiert wird und die genannten Zeichen hinter dem z eingeordnet sind. Leider bietet MySQL von Haus
aus noch keine deutsche Sortiervorschrift für UTF-8-Daten. Man kann sich aber mit einem Trick behelfen, indem die Daten für den Sortiervorgang in den gewünschten Zeichensatz mit der gewünschten
Sortiervorschrift konvertiert
werden:

```
‹?php

SELECT utf8_spalte FROM tabelle


ORDER BY CONVERT(utf8_spalte USING latin1) COLLATE latin1_german2_ci ASC

?›
```

Für die Sortierung werden hier die UTF-8-Werte in den Zeichensatz Latin-1 mit der Sortiervorschrift latin1_german2_ci konvertiert, die dafür sorgt, dass ä, ö, ü und ß bei der Sortierung wie ae, oe, ue und ss behandelt werden.

Beispiele

- Eine Seite mit ausschließlich chinesischem Inhalt (z.B. Zeichensatz GB2312) könnte folgendermaßen aufgebaut sein:

  ```
  ‹?php

  header(&amp;#39;Content-Type: text/html; charset=GB2312&amp;#39;);
  ```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="zh" lang="zh">

<head>

<title>GB2312-Example</title>

<meta http-equiv="Content-Type" content="text/html; charset=GB2312" />

<meta http-equiv="Content-Language" content="zh" />

<meta name="DC.language" scheme="DCTERMS.RFC3066" content="zh" />

<!-- weitere Angaben im Kopf... -->

</head>

<body>

<!-- beliebiger Inhalt, die chinesischen Schriftzeichen können direkt im

Quelltext notiert werden -->

/**

* jetzt noch irgendein dynamischer Inhalt

*/

$link = mysql_connect('host', 'user', 'pass')

or die('No Connection');

// wir wollen mit dem MySQL-Server (ab 4.1) nur noch gb2312 sprechen

mysql_query('SET NAMES \'gb2312\'', $link);

$result = mysql_query('SELECT gb2312_spalte FROM datenbank.tabelle...', $link);

// Ergebnisse ausgeben

</body>

</html>

?›

Eindeutige ID: #1058
Verfasser des Artikels: Kundensupport
Letzte Änderung des Artikels: 2012-07-20 08:07