Abstract

This is the MySQL Globalization extract from the MySQL 5.6 Reference Manual.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2022-04-07 (revision: 72703)
# Table of Contents

Preface and Legal Notices ........................................................................................................... v
1 Character Sets, Collations, Unicode ......................................................................................... 1

1.1 Character Sets and Collations in General ............................................................................... 2
1.2 Character Sets and Collations in MySQL ............................................................................... 3
  1.2.1 Character Set Repertoire ................................................................................................. 5
  1.2.2 UTF-8 for Metadata ......................................................................................................... 6
1.3 Specifying Character Sets and Collations ............................................................................... 8
  1.3.1 Collation Naming Conventions ....................................................................................... 8
  1.3.2 Server Character Set and Collation ............................................................................... 9
  1.3.3 Database Character Set and Collation .......................................................................... 10
  1.3.4 Table Character Set and Collation ............................................................................... 11
  1.3.5 Column Character Set and Collation ............................................................................ 11
  1.3.6 Character String Literal Character Set and Collation .................................................. 13
  1.3.7 The National Character Set ......................................................................................... 14
  1.3.8 Character Set Introducers .............................................................................................. 15
  1.3.9 Examples of Character Set and Collation Assignment .................................................. 16
  1.3.10 Compatibility with Other DBMSs .............................................................................. 17
1.4 Connection Character Sets and Collations .......................................................................... 17
1.5 Configuring Application Character Set and Collation .......................................................... 24
1.6 Error Message Character Set ............................................................................................... 25
1.7 Column Character Set Conversion ....................................................................................... 26

1.8 Collation Issues .................................................................................................................... 28
  1.8.1 Using COLLATE in SQL Statements .............................................................................. 28
  1.8.2 COLLATE Clause Precedence ....................................................................................... 28
  1.8.3 Character Set and Collation Compatibility ................................................................... 29
  1.8.4 Collation Coercibility in Expressions ........................................................................... 29
  1.8.5 The binary Collation Compared to _bin Collations ....................................................... 30
  1.8.6 Examples of the Effect of Collation .............................................................................. 32
  1.8.7 Using Collation in INFORMATION_SCHEMA Searches ................................................ 34
1.9 Unicode Support ................................................................................................................... 36
  1.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) .................................... 37
  1.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) .................................... 38
  1.9.3 The utf8 Character Set (Alias for utf8mb3) .................................................................. 38
  1.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding) ...................................................... 39
  1.9.5 The utf16 Character Set (UTF-16 Unicode Encoding) .................................................. 39
  1.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding) ........................................... 39
  1.9.7 The utf32 Character Set (UTF-32 Unicode Encoding) .................................................. 39
  1.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets .................................. 40
1.10 Supported Character Sets and Collations .......................................................................... 42
  1.10.1 Unicode Character Sets ............................................................................................... 43
  1.10.2 West European Character Sets .................................................................................... 49
  1.10.3 Central European Character Sets ................................................................................ 50
  1.10.4 South European and Middle East Character Sets ........................................................ 51
  1.10.5 Baltic Character Sets .................................................................................................... 52
  1.10.6 Cyrillic Character Sets .................................................................................................. 52
  1.10.7 Asian Character Sets .................................................................................................... 53
  1.10.8 The Binary Character Set ............................................................................................ 56
1.11 Restrictions on Character Sets ............................................................................................ 57
1.12 Setting the Error Message Language .................................................................................. 58
1.13 Adding a Character Set ........................................................................................................ 58
  1.13.1 Character Definition Arrays ........................................................................................ 60
1.13.2 String Collating Support for Complex Character Sets ........................................ 61
1.13.3 Multi-Byte Character Support for Complex Character Sets ................................ 61
1.14 Adding a Collation to a Character Set .................................................................... 62
  1.14.1 Collation Implementation Types ........................................................................ 63
  1.14.2 Choosing a Collation ID .................................................................................. 65
  1.14.3 Adding a Simple Collation to an 8-Bit Character Set ........................................ 66
  1.14.4 Adding a UCA Collation to a Unicode Character Set ....................................... 67
1.15 Character Set Configuration .................................................................................. 74
1.16 MySQL Server Locale Support .............................................................................. 75
2 MySQL Server Time Zone Support ............................................................................ 81
Preface and Legal Notices

This is the MySQL Globalization extract from the MySQL 5.6 Reference Manual.

**Licensing information—MySQL 5.6.** This product may include third-party software, used under license. If you are using a Commercial release of MySQL 5.6, see the MySQL 5.6 Commercial Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a Community release of MySQL 5.6, see the MySQL 5.6 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

**Licensing information—MySQL NDB Cluster 7.3.** This product may include third-party software, used under license. If you are using a Commercial release of NDB Cluster 7.3, see the MySQL NDB Cluster 7.3 Commercial Release License Information User Manual for licensing information relating to third-party software that may be included in this Commercial release. If you are using a Community release of NDB Cluster 7.3, see the MySQL NDB Cluster 7.3 Community Release License Information User Manual for licensing information relating to third-party software that may be included in this Community release.

**Licensing information—MySQL NDB Cluster 7.4.** This product may include third-party software, used under license. If you are using a Commercial release of NDB Cluster 7.4, see the MySQL NDB Cluster 7.4 Commercial Release License Information User Manual for licensing information relating to third-party software that may be included in this Commercial release. If you are using a Community release of NDB Cluster 7.4, see the MySQL NDB Cluster 7.4 Community Release License Information User Manual for licensing information relating to third-party software that may be included in this Community release.

**Legal Notices**

Copyright © 1997, 2022, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.
This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.
# Chapter 1 Character Sets, Collations, Unicode

Table of Contents

1.1 Character Sets and Collations in General ................................................. 2  
1.2 Character Sets and Collations in MySQL .................................................. 3  
  1.2.1 Character Set Repertoire .................................................................. 5  
  1.2.2 UTF-8 for Metadata ......................................................................... 6  
1.3 Specifying Character Sets and Collations .................................................... 8  
  1.3.1 Collation Naming Conventions .......................................................... 8  
  1.3.2 Server Character Set and Collation ................................................. 9  
  1.3.3 Database Character Set and Collation .......................................... 10  
  1.3.4 Table Character Set and Collation ............................................... 11  
  1.3.5 Column Character Set and Collation ........................................... 11  
  1.3.6 Character String Literal Character Set and Collation .................... 13  
  1.3.7 The National Character Set .......................................................... 14  
  1.3.8 Character Set Introducers .............................................................. 15  
  1.3.9 Examples of Character Set and Collation Assignment .................... 16  
  1.3.10 Compatibility with Other DBMSs ............................................... 17  
1.4 Connection Character Sets and Collations ................................................... 17  
1.5 Configuring Application Character Set and Collation .................................. 24  
1.6 Error Message Character Set .................................................................... 25  
1.7 Column Character Set Conversion .............................................................. 26  
1.8 Collation Issues ......................................................................................... 28  
  1.8.1 Using COLLATE in SQL Statements .............................................. 28  
  1.8.2 COLLATE Clause Precedence .......................................................... 28  
  1.8.3 Character Set and Collation Compatibility ................................... 29  
  1.8.4 Collation Coercibility in Expressions ......................................... 29  
  1.8.5 The binary Collation Compared to _bin Collations ......................... 30  
  1.8.6 Examples of the Effect of Collation ............................................... 32  
  1.8.7 Using Collation in INFORMATION_SCHEMA Searches ................... 34  
1.9 Unicode Support ......................................................................................... 36  
  1.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) .......... 37  
  1.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) ........ 38  
  1.9.3 The utf8 Character Set (Alias for utf8mb3) ................................... 38  
  1.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding) .......................... 39  
  1.9.5 The utf16 Character Set (UTF-16 Unicode Encoding) ...................... 39  
  1.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding) ............... 39  
  1.9.7 The utf32 Character Set (UTF-32 Unicode Encoding) ...................... 39  
  1.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets .......... 40  
1.10 Supported Character Sets and Collations ............................................... 42  
  1.10.1 Unicode Character Sets ................................................................ 43  
  1.10.2 West European Character Sets ...................................................... 49  
  1.10.3 Central European Character Sets ................................................... 50  
  1.10.4 South European and Middle East Character Sets ......................... 51  
  1.10.5 Baltic Character Sets ...................................................................... 52  
  1.10.6 Cyrillic Character Sets ................................................................... 52  
  1.10.7 Asian Character Sets ...................................................................... 53  
  1.10.8 The Binary Character Set .............................................................. 56  
1.11 Restrictions on Character Sets ................................................................. 57  
1.12 Setting the Error Message Language ......................................................... 58  
1.13 Adding a Character Set ............................................................................ 58
MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. The default MySQL server character set and collation are `latin1` and `latin1_swedish_ci`, but you can specify character sets at the server, database, table, column, and string literal levels.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment.
- Syntax for specifying character sets and collations.
- Affected functions and operations.
- Unicode support.
- The character sets and collations that are available, with notes.
- Selecting the language for error messages.
- Selecting the locale for day and month names.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you’ll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see Section 1.5, “Configuring Application Character Set and Collation”, and Section 1.4, “Connection Character Sets and Collations”.

## 1.1 Character Sets and Collations in General

A **character set** is a set of symbols and encodings. A **collation** is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: `A`, `B`, `a`, `b`. We give each letter a number: `A = 0`, `B = 1`, `a = 2`, `b = 3`. The letter `A` is a symbol, the number 0 is the encoding for `A`, and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, `A` and `B`. The simplest way to do this is to look at the encodings: 0 for `A` and 1 for `B`. Because 0 is less than 1, we say `A` is less than `B`. What we've just done is
apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a binary collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters $a$ and $b$ as equivalent to $A$ and $B$; (2) then compare the encodings. We call this a case-insensitive collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just $A$ and $B$ but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German Ò), and for multiple-character mappings (such as the rule that Ò = OE in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets.
- Compare strings using a variety of collations.
- Mix strings with different character sets or collations in the same server, the same database, or even the same table.
- Enable specification of character set and collation at any level.

To use these features effectively, you must know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

## 1.2 Character Sets and Collations in MySQL

MySQL Server supports multiple character sets. To display the available character sets, use the INFORMATION_SCHEMA CHARACTER_SETS table or the SHOW CHARACTER SET statement. A partial listing follows. For more complete information, see Section 1.10, “Supported Character Sets and Collations”.

<table>
<thead>
<tr>
<th>Charset</th>
<th>Description</th>
<th>Default collation</th>
<th>Maxlen</th>
</tr>
</thead>
<tbody>
<tr>
<td>big5</td>
<td>Big5 Traditional Chinese</td>
<td>big5_chinese_ci</td>
<td>2</td>
</tr>
<tr>
<td>latin1</td>
<td>cp1252 West European</td>
<td>latin1_swedish_ci</td>
<td>1</td>
</tr>
<tr>
<td>latin2</td>
<td>ISO 8859-2 Central European</td>
<td>latin2_general_ci</td>
<td>1</td>
</tr>
<tr>
<td>utf8</td>
<td>UTF-8 Unicode</td>
<td>utf8_general_ci</td>
<td>3</td>
</tr>
<tr>
<td>ucs2</td>
<td>UCS-2 Unicode</td>
<td>ucs2_general_ci</td>
<td>2</td>
</tr>
<tr>
<td>utf8mb4</td>
<td>UTF-8 Unicode</td>
<td>utf8mb4_general_ci</td>
<td>4</td>
</tr>
<tr>
<td>binary</td>
<td>Binary pseudo charset</td>
<td>binary</td>
<td>1</td>
</tr>
</tbody>
</table>

By default, the SHOW CHARACTER SET statement displays all available character sets. It takes an optional LIKE or WHERE clause that indicates which character set names to match. For example:

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

<table>
<thead>
<tr>
<th>Charset</th>
<th>Description</th>
<th>Default collation</th>
<th>Maxlen</th>
</tr>
</thead>
<tbody>
<tr>
<td>latin1</td>
<td>cp1252 West European</td>
<td>latin1_swedish_ci</td>
<td>1</td>
</tr>
<tr>
<td>latin2</td>
<td>ISO 8859-2 Central European</td>
<td>latin2_general_ci</td>
<td>1</td>
</tr>
</tbody>
</table>
A given character set always has at least one collation, and most character sets have several. To list the display collations for a character set, use the INFORMATION_SCHEMA COLLATIONS table or the SHOW COLLATION statement.

By default, the SHOW COLLATION statement displays all available collations. It takes an optional LIKE or WHERE clause that indicates which collation names to display. For example, to see the collations for the default character set, latin1 (cp1252 West European), use this statement:

```
mysql> SHOW COLLATION WHERE Charset = 'latin1';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci  | latin1  | 15 |         | Yes      |       1 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         | Yes      |       1 |
| latin1_general_cs | latin1  | 49 |         | Yes      |       1 |
| latin1_spanish_ci | latin1  | 94 |         | Yes      |       1 |
+-------------------+---------+----+---------+----------+---------+
```

The latin1 collations have the following meanings.

<table>
<thead>
<tr>
<th>Collation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>latin1_bin</td>
<td>Binary according to latin1 encoding</td>
</tr>
<tr>
<td>latin1_danish_ci</td>
<td>Danish/Norwegian</td>
</tr>
<tr>
<td>latin1_general_ci</td>
<td>Multilingual (Western European)</td>
</tr>
<tr>
<td>latin1_general_cs</td>
<td>Multilingual (ISO Western European), case-sensitive</td>
</tr>
<tr>
<td>latin1_german1_ci</td>
<td>German DIN-1 (dictionary order)</td>
</tr>
<tr>
<td>latin1_german2_ci</td>
<td>German DIN-2 (phone book order)</td>
</tr>
<tr>
<td>latin1_spanish_ci</td>
<td>Modern Spanish</td>
</tr>
<tr>
<td>latin1_swedish_ci</td>
<td>Swedish/Finnish</td>
</tr>
</tbody>
</table>

Collations have these general characteristics:

- Two different character sets cannot have the same collation.

- Each character set has a default collation. For example, the default collations for latin1 and utf8 are latin1_swedish_ci and utf8_general_ci, respectively. The INFORMATION_SCHEMA CHARACTER_SETS table and the SHOW CHARACTER SET statement indicate the default collation for each character set. The INFORMATION_SCHEMA COLLATIONS table and the SHOW COLLATION statement have a column that indicates for each collation whether it is the default for its character set (Yes if so, empty if not).

- Collation names start with the name of the character set with which they are associated, generally followed by one or more suffixes indicating other collation characteristics. For additional information about naming conventions, see Section 1.3.1, “Collation Naming Conventions”.

When a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing an inappropriate collation, perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.
1.2.1 Character Set Repertoire

The *repertoire* of a character set is the collection of characters in the set.

String expressions have a repertoire attribute, which can have two values:

- **ASCII**: The expression can contain only ASCII characters; that is, characters in the Unicode range \u0000 to \u007F.

- **UNICODE**: The expression can contain characters in the Unicode range \u0000 to \u10FFFF. This includes characters in the Basic Multilingual Plane (BMP) range (\u0000 to \uFFFF) and supplementary characters outside the BMP range (\u10000 to \u10FFFF).

The ASCII range is a subset of UNICODE range, so a string with ASCII repertoire can be converted safely without loss of information to the character set of any string with UNICODE repertoire. It can also be converted safely to any character set that is a superset of the ascii character set. (All MySQL character sets are supersets of ascii with the exception of swe7, which reuses some punctuation characters for Swedish accented characters.)

The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an "illegal mix of collations" error when the rules for collation coercibility are insufficient to resolve ambiguities. (For information about coercibility, see Section 1.8.4, "Collation Coercibility in Expressions").

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for a string constant depends on string content and may differ from the repertoire of the string character set. Consider these statements:

  ```sql
  SET NAMES utf8; SELECT 'abc';
  SELECT _utf8'def';
  SELECT N'MySQL';
  ```

  Although the character set is utf8 in each of the preceding cases, the strings do not actually contain any characters outside the ASCII range, so their repertoire is ASCII rather than UNICODE.

- A column having the ascii character set has ASCII repertoire because of its character set. In the following table, c1 has ASCII repertoire:

  ```sql
  CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
  ```

  The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

  ```sql
  CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET latin1,
                   c2 CHAR(1) CHARACTER SET ascii);
  INSERT INTO t1 VALUES ('a','b');
  SELECT CONCAT(c1,c2) FROM t1;
  ```

  Without repertoire, this error occurs:

  ```sql
  ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
  and (ascii_general_ci,IMPLICIT) for operation 'concat'
  ```

  Using repertoire, subset to superset (ascii to latin1) conversion can occur and a result is returned:
• Functions with one string argument inherit the repertoire of their argument. The result of \( \text{UPPER(_utf8'abc')} \) has ASCII repertoire because its argument has ASCII repertoire. (Despite the _utf8 introducer, the string 'abc' contains no characters outside the ASCII range.)

• For functions that return a string but do not have string arguments and use character_set_connection as the result character set, the result repertoire is ASCII if character_set_connection is ascii, and UNICODE otherwise:

\[
\text{FORMAT(numeric_column, 4)};
\]

Use of repertoire changes how MySQL evaluates the following example:

```sql
SET NAMES ascii;
CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
INSERT INTO t1 VALUES (1,'b');
SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE) and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-------------------------+
| CONCAT(FORMAT(a, 4), b) |
| 1.0000b                |
+-------------------------+
```

• Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire, where UNICODE is wider than ASCII. Consider the following CONCAT() calls:

\[
\text{CONCAT(_ucs2 X'0041', _ucs2 X'0042')}\]
\[
\text{CONCAT(_ucs2 X'0041', _ucs2 X'00C2')}\]

For the first call, the repertoire is ASCII because both arguments are within the ASCII range. For the second call, the repertoire is UNICODE because the second argument is outside the ASCII range.

• The repertoire for function return values is determined based on the repertoire of only those arguments that affect the result's character set and collation.

\[
\text{IF(column1 < column2, 'smaller', 'greater')}\]

The result repertoire is ASCII because the two string arguments (the second argument and the third argument) both have ASCII repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

1.2.2 UTF-8 for Metadata

Metadata is “the data about the data.” Anything that describes the database—as opposed to being the contents of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from SHOW are metadata. This is also true of the contents of tables in INFORMATION_SCHEMA because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:
• All metadata must be in the same character set. Otherwise, neither the `SHOW` statements nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.

• Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()`, `CURRENT_USER()`, `SESSION_USER()`, `SYSTEM_USER()`, `DATABASE()`, and `VERSION()` functions have the UTF-8 character set by default.

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>character_set_system</td>
<td>utf8</td>
</tr>
</tbody>
</table>
+----------------------|-------+
```

Storage of metadata using Unicode does not mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` system variable, which has a default value of `utf8`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` and other related system variables. (See Section 1.4, “Connection Character Sets and Collations”.) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client to perform the conversion, but this option is not always available for all clients.

If `character_set_results` is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment.

Although automatic conversion is not in the SQL standard, the standard does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for
comparisons with non-Unicode strings. For more information about coercion of strings, see Section 1.8.4, “Collation Coercibility in Expressions”.

1.3 Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

**CHARACTER SET** is used in clauses that specify a character set. **CHARSET** can be used as a synonym for **CHARACTER SET**.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you’ll need to indicate which one. For example, to use the **utf8** Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see Section 1.4, “Connection Character Sets and Collations”.

1.3.1 MySQL Naming Conventions

MySQL collation names follow these conventions:

- A collation name starts with the name of the character set with which it is associated, generally followed by one or more suffixes indicating other collation characteristics. For example, **utf8_general_ci** and **latin1_swedish_ci** are collations for the **utf8** and **latin1** character sets, respectively. The **binary** character set has a single collation, also named **binary**, with no suffixes.

- A language-specific collation includes a language name. For example, **utf8_turkish_ci** and **utf8_hungarian_ci** sort characters for the **utf8** character set using the rules of Turkish and Hungarian, respectively.

- Collation suffixes indicate whether a collation is case-sensitive, accent-sensitive, or kana-sensitive (or some combination thereof), or binary. The following table shows the suffixes used to indicate these characteristics.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>_ai</td>
<td>Accent-insensitive</td>
</tr>
<tr>
<td>_as</td>
<td>Accent-sensitive</td>
</tr>
<tr>
<td>_ci</td>
<td>Case-insensitive</td>
</tr>
<tr>
<td>_cs</td>
<td>Case-sensitive</td>
</tr>
<tr>
<td>_bin</td>
<td>Binary</td>
</tr>
</tbody>
</table>

For nonbinary collation names that do not specify accent sensitivity, it is determined by case sensitivity. If a collation name does not contain _ai or _as, _ci in the name implies _ai and _cs in the name implies _as. For example, **latin1_general_ci** is explicitly case insensitive and implicitly accent-insensitive, and **latin1_general_cs** is explicitly case-sensitive and implicitly accent-sensitive.

For the **binary** collation of the **binary** character set, comparisons are based on numeric byte values. For the _bin collation of a nonbinary character set, comparisons are based on numeric character code...
Server Character Set and Collation

values, which differ from byte values for multibyte characters. For information about the differences between the binary collation of the binary character set and the _bin collations of nonbinary character sets, see Section 1.8.5, “The binary Collation Compared to _bin Collations”.

• Collation names for Unicode character sets may include a version number to indicate the version of the Unicode Collation Algorithm (UCA) on which the collation is based. UCA-based collations without a version number in the name use the version-4.0.0 UCA weight keys. For example:

  • utf8_unicode_520_ci is based on UCA 5.2.0 weight keys (http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt).
  • utf8_unicode_ci (with no version named) is based on UCA 4.0.0 weight keys (http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt).
  • For Unicode character sets, the xxx_general_mysql500_ci collations preserve the pre-5.1.24 ordering of the original xxx_general_ci collations and permit upgrades for tables created before MySQL 5.1.24 (Bug #27877).

1.3.2 Server Character Set and Collation

MySQL Server has a server character set and a server collation. By default, these are latin1 and latin1_swedish_ci, but they can be set explicitly at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start mysqld. You can use --character-set-server for the character set. Along with it, you can add --collation-server for the collation. If you don't specify a character set, that is the same as saying --character-set-server=latin1. If you specify only a character set (for example, latin1) but not a collation, that is the same as saying --character-set-server=latin1 --collation-server=latin1_swedish_ci because latin1_swedish_ci is the default collation for latin1. Therefore, the following three commands all have the same effect:

  mysql
  mysqld --character-set-server=latin1
  mysqld --character-set-server=latin1 --collation=latin1_swedish_ci

One way to change the settings is by recompiling. To change the default server character set and collation when building from sources, use the DEFAULT_CHARSET and DEFAULT_COLLATION options for CMake. For example:

  cmake . -DDEFAULT_CHARSET=latin1

Or:

  cmake . -DDEFAULT_CHARSET=latin1
   -DDEFAULT_COLLATION=latin1_german1_ci

Both mysqld and CMake verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in CREATE DATABASE statements. They have no other purpose.

The current server character set and collation can be determined from the values of the character_set_server and collation_server system variables. These variables can be changed at runtime.
1.3.3 Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

The keyword `SCHEMA` can be used instead of `DATABASE`.

All database options are stored in a text file named `db.opt` that can be found in the database directory.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used.
- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set `charset_name` and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.
- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with `collation_name` and collation `collation_name` are used.
- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the server character set and server collation are used.

The character set and collation for the default database can be determined from the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` and `collation_server`.

To see the default character set and collation for a given database, use these statements:

```
USE db_name;
SELECT @@character_set_database, @@collation_database;
```

Alternatively, to display the values without changing the default database:

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'db_name';
```

The database character set and collation affect these aspects of server operation:

- For `CREATE TABLE` statements, the database character set and collation are used as default values for table definitions if the table character set and collation are not specified. To override this, provide explicit `CHARACTER SET` and `COLLATE` table options.
• For **LOAD DATA** statements that include no **CHARACTER SET** clause, the server uses the character set indicated by the **character_set_database** system variable to interpret the information in the file. To override this, provide an explicit **CHARACTER SET** clause.

• For stored routines (procedures and functions), the database character set and collation in effect at routine creation time are used as the character set and collation of character data parameters for which the declaration includes no **CHARACTER SET** or a **COLLATE** attribute. To override this, provide **CHARACTER SET** and **COLLATE** explicitly.

### 1.3.4 Table Character Set and Collation

Every table has a table character set and a table collation. The **CREATE TABLE** and **ALTER TABLE** statements have optional clauses for specifying the table character set and collation:

```sql
CREATE TABLE tbl_name (column_list)
[[DEFAULT] CHARACTER SET charset_name]
[[COLLATE collation_name]]
ALTER TABLE tbl_name
[[DEFAULT] CHARACTER SET charset_name]
[[COLLATE collation_name]]
```

Example:

```sql
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

• If both **CHARACTER SET charset_name** and **COLLATE collation_name** are specified, character set **charset_name** and collation **collation_name** are used.

• If **CHARACTER SET charset_name** is specified without **COLLATE**, character set **charset_name** and its default collation are used. To see the default collation for each character set, use the **SHOW CHARACTER SET** statement or query the **INFORMATION_SCHEMA CHARACTER_SETS** table.

• If **COLLATE collation_name** is specified without **CHARACTER SET**, the character set associated with **collation_name** and collation **collation_name** are used.

• Otherwise (neither **CHARACTER SET** nor **COLLATE** is specified), the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

### 1.3.5 Column Character Set and Collation

Every “character” column (that is, a column of type **CHAR**, **VARCHAR**, a **TEXT** type, or any synonym) has a column character set and a column collation. Column definition syntax for **CREATE TABLE** and **ALTER TABLE** has optional clauses for specifying the column character set and collation:

```sql
col_name (CHAR | VARCHAR | TEXT) (col_length)
[CHARACTER SET charset_name]
[COLLATE collation_name]
```

These clauses can also be used for **ENUM** and **SET** columns:

```sql
col_name (ENUM | set) (val_list)
[CHARACTER SET charset_name]
[COLLATE collation_name]
```
Column Character Set and Collation

Examples:

```
CREATE TABLE t1
(
  col1 VARCHAR(5)
  CHARACTER SET latin1
  COLLATE latin1_german1_ci
);
ALTER TABLE t1 MODIFY
  col1 VARCHAR(5)
  CHARACTER SET latin1
  COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- **If both CHARACTER SET charset_name and COLLATE collation_name are specified, character set charset_name and collation collation_name are used.**

```
CREATE TABLE t1
(
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set and collation are specified for the column, so they are used. The column has character set utf8 and collation utf8_unicode_ci.

- **If CHARACTER SET charset_name is specified without COLLATE, character set charset_name and its default collation are used.**

```
CREATE TABLE t1
(
  col1 CHAR(10) CHARACTER SET utf8
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The character set is specified for the column, but the collation is not. The column has character set utf8 and the default collation for utf8, which is utf8_general_ci. To see the default collation for each character set, use the SHOW CHARACTER SET statement or query the INFORMATION_SCHEMA CHARACTER_SETS table.

- **If COLLATE collation_name is specified without CHARACTER SET, the character set associated with collation_name and collation collation_name are used.**

```
CREATE TABLE t1
(
  col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The collation is specified for the column, but the character set is not. The column has collation utf8_polish_ci and the character set is the one associated with the collation, which is utf8.

- **Otherwise (neither CHARACTER SET nor COLLATE is specified), the table character set and collation are used.**

```
CREATE TABLE t1
(
  col1 CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_bin;
```

Neither the character set nor collation is specified for the column, so the table defaults are used. The column has character set latin1 and collation latin1_bin.

The CHARACTER SET and COLLATE clauses are standard SQL.
If you use \texttt{ALTER TABLE} to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

### 1.3.6 Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

For the simple statement \texttt{SELECT 'string'}, the string has the connection default character set and collation defined by the \texttt{character_set_connection} and \texttt{collation_connection} system variables.

A character string literal may have an optional character set introducer and \texttt{COLLATE} clause, to designate it as a string that uses a particular character set and collation:

\[ \texttt{[\_charset_name]'string' [COLLATE \texttt{collation_name}]} \]

The \texttt{\_charset_name} expression is formally called an \textit{introducer}. It tells the parser, “the string that follows uses character set \texttt{\_charset_name}.” An introducer does not change the string to the introducer character set like \texttt{CONVERT()} would do. It does not change the string value, although padding may occur. The introducer is just a signal. See Section 1.3.8, “Character Set Introducers”.

Examples:

\begin{verbatim}
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8'abc' COLLATE utf8_danish_ci;
\end{verbatim}

Character set introducers and the \texttt{COLLATE} clause are implemented according to standard SQL specifications.

MySQL determines the character set and collation of a character string literal in the following manner:

- If both \texttt{\_charset_name} and \texttt{COLLATE \texttt{collation_name}} are specified, character set \texttt{\_charset_name} and collation \texttt{\texttt{collation_name}} are used.\texttt{\texttt{collation_name}} must be a permitted collation for \texttt{\_charset_name}.
- If \texttt{\_charset_name} is specified but \texttt{COLLATE} is not specified, character set \texttt{\_charset_name} and its default collation are used. To see the default collation for each character set, use the \texttt{SHOW CHARACTER SET} statement or query the \texttt{INFORMATION_SCHEMA CHARACTER SETS} table.
- If \texttt{\_charset_name} is not specified but \texttt{\texttt{COLLATE \texttt{collation_name}}} is specified, the connection default character set given by the \texttt{\texttt{character_set_connection}} system variable and collation \texttt{\texttt{collation_name}} are used. \texttt{\texttt{collation_name}} must be a permitted collation for the connection default character set.
- Otherwise (neither \texttt{\_charset_name} nor \texttt{\texttt{COLLATE \texttt{collation_name}}} is specified), the connection default character set and collation given by the \texttt{\texttt{character_set_connection}} and \texttt{\texttt{collation_connection}} system variables are used.

Examples:

- A nonbinary string with \texttt{\texttt{latin1}} character set and \texttt{\texttt{latin1\_german1\_ci}} collation:
  \begin{verbatim}
  SELECT _\texttt{latin1}'Müller' COLLATE latin1\_german1\_ci;
  \end{verbatim}
- A nonbinary string with \texttt{\texttt{utf8}} character set and its default collation (that is, \texttt{\texttt{utf8\_general\_ci}}):
  \begin{verbatim}
  SELECT _\texttt{utf8}'Müller';
  \end{verbatim}
- A binary string with \texttt{\texttt{binary}} character set and its default collation (that is, \texttt{\texttt{binary}}):
  \begin{verbatim}
  SELECT _\texttt{binary}'Müller';
  \end{verbatim}
The National Character Set

SELECT _binary'Müller';

- A nonbinary string with the connection default character set and `utf8_general_ci` collation (fails if the connection character set is not `utf8`):

  SELECT 'Müller' COLLATE utf8_general_ci;

- A string with the connection default character set and collation:

  SELECT 'Müller';

An introducer indicates the character set for the following string, but does not change how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`.

The following examples show that escape processing occurs using `character_set_connection` even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection`, as discussed in Section 1.4, “Connection Character Sets and Collations”), and display the resulting strings using the `HEX()` function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+------------+-----------------+
<table>
<thead>
<tr>
<th>HEX('à\n')</th>
<th>HEX(_sjis'à\n')</th>
</tr>
</thead>
<tbody>
<tr>
<td>E00A</td>
<td>E00A</td>
</tr>
</tbody>
</table>
+------------+-----------------+
```

Here, à (hexadecimal value E0) is followed by \n, the escape sequence for newline. The escape sequence is interpreted using the `character_set_connection` value of latin1 to produce a literal newline (hexadecimal value 0A). This happens even for the second string. That is, the _sjis introducer does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+------------+-------------------+
<table>
<thead>
<tr>
<th>HEX('à\n')</th>
<th>HEX(_latin1'à\n')</th>
</tr>
</thead>
<tbody>
<tr>
<td>E05C6E</td>
<td>E05C6E</td>
</tr>
</tbody>
</table>
+------------+--------------------+
```

Here, `character_set_connection` is sjis, a character set in which the sequence of à followed by \ (hexadecimal values 05 and 5C) is a valid multibyte character. Hence, the first two bytes of the string are interpreted as a single sjis character, and the \ is not interpreted as an escape character. The following n (hexadecimal value 6E) is not interpreted as part of an escape sequence. This is true even for the second string; the _latin1 introducer does not affect escape processing.

1.3.7 The National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
```
As are these:

```
NCHAR(10)
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NVARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

### 1.3.8 Character Set Introducers

A character string literal, hexadecimal literal, or bit-value literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_[charset_name] ] literal [COLLATE collation_name]
```

The `_[charset_name]` expression is formally called an *introducer*. It tells the parser, “the string that follows uses character set `charset_name`.” An introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string value, although padding may occur. The introducer is just a signal.

For character string literals, space between the introducer and the string is permitted but optional.

For character set literals, an introducer indicates the character set for the following string, but does not change how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`. For additional discussion and examples, see Section 1.3.6, “Character String Literal Character Set and Collation”.

Examples:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8'abc' COLLATE utf8_danish_ci;
SELECT _latin1 X'4D7953514C';
SELECT _utf8 0x4D7953514C COLLATE utf8_danish_ci;
SELECT _latin1 b'1000001';
SELECT _utf8 0b1000001 COLLATE utf8_danish_ci;
```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

Character string literals can be designated as binary strings by using the `_binary` introducer. Hexadecimal literals and bit-value literals are binary strings by default, so `_binary` is permitted, but unnecessary.

MySQL determines the character set and collation of a character string literal, hexadecimal literal, or bit-value literal in the following manner:

1. If both `_[charset_name]` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used. `collation_name` must be a permitted collation for `charset_name`.

2. If only the `COLLATE` clause is specified, the collation is the default collation for the character set.

3. If only the `_[charset_name]` introducer is specified, the character set is the character set specified by `character_set_connection`.
Examples of Character Set and Collation Assignment

- If \_charset\_name is specified but COLLATE is not specified, character set \_charset\_name and its default collation are used. To see the default collation for each character set, use the SHOW CHARACTER SET statement or query the INFORMATION_SCHEMA CHARACTER_SETS table.

- If \_charset\_name is not specified but COLLATE \_collation\_name is specified:
  - For a character string literal, the connection default character set given by the character\_set\_connection system variable and collation \_collation\_name are used. \_collation\_name must be a permitted collation for the connection default character set.
  - For a hexadecimal literal or bit-value literal, the only permitted collation is binary because these types of literals are binary strings by default.

- Otherwise (neither \_charset\_name nor COLLATE \_collation\_name is specified):
  - For a character string literal, the connection default character set and collation given by the character\_set\_connection and collation\_connection system variables are used.
  - For a hexadecimal literal or bit-value literal, the character set and collation are binary.

Examples:

- Nonbinary strings with latin1 character set and latin1\_german1\_ci collation:
  ```sql
  SELECT \_latin1'Müller' COLLATE latin1\_german1\_ci;
  SELECT \_latin1 X'0A0D' COLLATE latin1\_german1\_ci;
  SELECT \_latin1 b'0110' COLLATE latin1\_german1\_ci;
  ```

- Nonbinary strings with utf8 character set and its default collation (that is, utf8\_general\_ci):
  ```sql
  SELECT \_utf8'Müller';
  SELECT \_utf8 X'0A0D';
  SELECT \_utf8 b'0110';
  ```

- Binary strings with binary character set and its default collation (that is, binary):
  ```sql
  SELECT \_binary'Müller';
  SELECT X'0A0D';
  SELECT b'0110';
  ```

The hexadecimal literal and bit-value literal need no introducer because they are binary strings by default.

- A nonbinary string with the connection default character set and utf8\_general\_ci collation (fails if the connection character set is not utf8):
  ```sql
  SELECT 'Müller' COLLATE utf8\_general\_ci;
  ```

This construction (COLLATE only) does not work for hexadecimal literals or bit literals because their character set is binary no matter the connection character set, and binary is not compatible with the utf8\_general\_ci collation. The only permitted COLLATE clause in the absence of an introducer is COLLATE binary.

- A string with the connection default character set and collation:
  ```sql
  SELECT 'Müller';
  ```

1.3.9 Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.
Example 1: Table and Column Definition

```
CREATE TABLE t1  
(   
c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci   
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1  
(   
c1 CHAR(10) CHARACTER SET latin1   
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1  
(   
c1 CHAR(10)   
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1   
  DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1  
(   
c1 CHAR(10)   
);
```

We create a column without specifying its character set and collation. We’re also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

1.3.10 Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

1.4 Connection Character Sets and Collations

A “connection” is what a client program makes when it connects to the server, to begin a session within which it interacts with the server. The client sends SQL statements, such as queries, over the session
connection. The server sends responses, such as result sets or error messages, over the connection back to the client.

- Connection Character Set and Collation System Variables
- Impermissible Client Character Sets
- Client Program Connection Character Set Configuration
- SQL Statements for Connection Character Set Configuration
- Connection Character Set Error Handling

## Connection Character Set and Collation System Variables

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The `character_set_server` and `collation_server` system variables indicate the server character set and collation. See Section 1.3.2, “Server Character Set and Collation”.

- The `character_set_database` and `collation_database` system variables indicate the character set and collation of the default database. See Section 1.3.3, “Database Character Set and Collation”.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has session-specific connection-related character set and collation system variables. These session system variable values are initialized at connect time, but can be changed within the session.

Several questions about character set and collation handling for client connections can be answered in terms of system variables:

- What character set are statements in when they leave the client?

  The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

  **Note**
  Some character sets cannot be used as the client character set. See Impermissible Client Character Sets.

- What character set should the server translate statements to after receiving them?

  To determine this, the server uses the `character_set_connection` and `collation_connection` system variables:

  - The server converts statements sent by the client from `character_set_client` to `character_set_connection`. Exception: For string literals that have an introducer such as `_utf8mb4` or `_latin2`, the introducer determines the character set. See Section 1.3.8, “Character Set Introducers”.

  - `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence (see Section 1.8.4, “Collation Coercibility in Expressions”).

- What character set should the server translate query results to before shipping them back to the client?
The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, result metadata such as column names, and error messages.

To tell the server to perform no conversion of result sets or error messages, set `character_set_results` to `NULL` or `binary`:

```sql
SET character_set_results = NULL;
SET character_set_results = binary;
```

For more information about character sets and error messages, see Section 1.6, “Error Message Character Set”.

To see the values of the character set and collation system variables that apply to the current session, use this statement:

```sql
SELECT * FROM INFORMATION_SCHEMA.SESSION_VARIABLES
WHERE VARIABLE_NAME IN ('character_set_client', 'character_set_connection',
'character_set_results', 'collation_connection') ORDER BY VARIABLE_NAME;
```

The following simpler statements also display the connection variables, but include other related variables as well. They can be useful to see all character set and collation system variables:

```sql
SHOW SESSION VARIABLES LIKE 'character\_%';
SHOW SESSION VARIABLES LIKE 'collation\_';
```

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings for each connection to the server.

**Impermissible Client Character Sets**

The `character_set_client` system variable cannot be set to certain character sets:

- `ucs2`
- `utf16`
- `utf16le`
- `utf32`

Attempting to use any of those character sets as the client character set produces an error:

```sql
mysql> SET character_set_client = 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client'
can't be set to the value of 'ucs2'
```

The same error occurs if any of those character sets are used in the following contexts, all of which result in an attempt to set `character_set_client` to the named character set:

- The `--default-character-set=charset_name` command option used by MySQL client programs such as `mysql` and `mysqladmin`.
- The `SET NAMES 'charset_name'` statement.
- The `SET CHARACTER SET 'charset_name'` statement.

**Client Program Connection Character Set Configuration**

When a client connects to the server, it indicates which character set it wants to use for communication with the server. (Actually, the client indicates the default collation for that character set, from...
which the server can determine the character set.) The server uses this information to set the
character_set_client, character_set_results, character_set_connection system
variables to the character set, and collation_connection to the character set default collation. In
effect, the server performs the equivalent of a SET NAMES operation.

If the server does not support the requested character set or collation, it falls back to using the server
character set and collation to configure the connection. For additional detail about this fallback behavior,
see Connection Character Set Error Handling.

The mysql, mysqladmin, mysqlcheck, mysqlimport, and mysqlshow client programs determine the
default character set to use as follows:

• In the absence of other information, each client uses the compiled-in default character set, usually
  latin1.

• Each client can autodetect which character set to use based on the operating system setting, such as
  the value of the LANG or LC_ALL locale environment variable on Unix systems or the code page setting
  on Windows systems. For systems on which the locale is available from the OS, the client uses it to
  set the default character set rather than using the compiled-in default. For example, setting LANG to
  ru_RU.KOI8-R causes the koi8r character set to be used. Thus, users can configure the locale in
  their environment for use by MySQL clients.

The OS character set is mapped to the closest MySQL character set if there is no exact match. If the
client does not support the matching character set, it uses the compiled-in default. For example, ucs2 is
not supported as a connection character set, so it maps to the compiled-in default.

C applications can use character set autodetection based on the OS setting by invoking
mysql_options() as follows before connecting to the server:

```c
mysql_options(mysql,
             MYSQL_SET_CHARSET_NAME,
             MYSQL_AUTODETECT_CHARSET_NAME);
```

• Each client supports a --default-character-set option, which enables users to specify the
  character set explicitly to override whatever default the client otherwise determines.

  **Note**

  Some character sets cannot be used as the client character set. Attempting
  to use them with --default-character-set produces an error. See
  Impermissible Client Character Sets.

With the mysql client, to use a character set different from the default, you could explicitly execute a SET
NAMES statement every time you connect to the server (see Client Program Connection Character Set
Configuration). To accomplish the same result more easily, specify the character set in your option file.
For example, the following option file setting changes the three connection-related character set system
variables set to koi8r each time you invoke mysql:

```plaintext
[mysql]
default-character-set=koi8r
```

If you are using the mysql client with auto-reconnect enabled (which is not recommended), it is preferable
use the charset command rather than SET NAMES. For example:

```mysql
mysql> charset koi8r
 charset changed
```

The charset command issues a SET NAMES statement, and also changes the default character set that
mysql uses when it reconnects after the connection has dropped.
When configuration client programs, you must also consider the environment within which they execute. See Section 1.5, "Configuring Application Character Set and Collation".

**SQL Statements for Connection Character Set Configuration**

After a connection has been established, clients can change the character set and collation system variables for the current session. These variables can be changed individually using `SET` statements, but two more convenient statements affect the connection-related character set system variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client uses to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, “future incoming messages from this client are in character set cp1251.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement that produces a result set.)

  A `SET NAMES 'charset_name'` statement is equivalent to these three statements:

  ```
  SET character_set_client = charset_name;
  SET character_set_results = charset_name;
  SET character_set_connection = charset_name;
  ```

  Setting `character_set_connection` to `charset_name` also implicitly sets `collation_connection` to the default collation for `charset_name`. It is unnecessary to set that collation explicitly. To specify a particular collation to use for `collation_connection`, add a `COLLATE` clause:

  ```
  SET NAMES 'charset_name' COLLATE 'collation_name'
  ```

- `SET CHARACTER SET 'charset_name'`

  `SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` and `collation_connection` to `character_set_database` and `collation_database` (which, as mentioned previously, indicate the character set and collation of the default database).

  A `SET CHARACTER SET charset_name` statement is equivalent to these three statements:

  ```
  SET character_set_client = charset_name;
  SET character_set_results = charset_name;
  SET collation_connection = @@collation_database;
  ```

  Setting `collation_connection` also implicitly sets `character_set_connection` to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is unnecessary to set `character_set_connection` explicitly.

  **Note**

  Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` or `SET CHARACTER SET` produces an error. See [Impermissible Client Character Sets](#).

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET 'latin1'` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy for characters that are not in both character sets.
Connection Character Set Error Handling

Attempts to use an inappropriate connection character set or collation can produce an error, or cause the server to fall back to its default character set and collation for a given connection. This section describes problems that can occur when configuring the connection character set. These problems can occur when establishing a connection or when changing the character set within an established connection.

- Connect-Time Error Handling
- Runtime Error Handling

Connect-Time Error Handling

Some character sets cannot be used as the client character set; see Impermissible Client Character Sets. If you specify a character set that is valid but not permitted as a client character set, the server returns an error:

```
$> mysql --default-character-set=ucs2
ERROR 1231 (42000): Variable 'character_set_client' can't be set to the value of 'ucs2'
```

If you specify a character set that the client does not recognize, it produces an error:

```
$> mysql --default-character-set=bogus
mysql: Character set 'bogus' is not a compiled character set and is not specified in the '/usr/local/mysql/share/charsets/Index.xml' file
ERROR 2019 (HY000): Can't initialize character set bogus (path: /usr/local/mysql/share/charsets/)
```

If you specify a character set that the client recognizes but the server does not, the server falls back to its default character set and collation. Suppose that the server is configured to use latin1 and latin1_swedish_ci as its defaults, and that it does not recognize gb18030 as a valid character set. A client that specifies `--default-character-set=gb18030` is able to connect to the server, but the resulting character set is not what the client wants:

```
mysql> SHOW SESSION VARIABLES LIKE 'character_set\_';
+--------------------------+--------+
| Variable_name            | Value  |
|--------------------------+--------+
| character_set_client     | latin1 |
| character_set_connection | latin1 |
| ...                      |        |
| character_set_results    | latin1 |
| ...                      |        |
mysql> SHOW SESSION VARIABLES LIKE 'collation_connection';
+----------------------+
<table>
<thead>
<tr>
<th>Variable_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>collation_connection</td>
</tr>
</tbody>
</table>
```

You can see that the connection system variables have been set to reflect a character set and collation of latin1 and latin1_swedish_ci. This occurs because the server cannot satisfy the client character set request and falls back to its defaults.

In this case, the client cannot use the character set that it wants because the server does not support it. The client must either be willing to use a different character set, or connect to a different server that supports the desired character set.

The same problem occurs in a more subtle context: When the client tells the server to use a character set that the server recognizes, but the default collation for that character set on the client side is not known.
Connection Character Set Error Handling

on the server side. This occurs, for example, when a MySQL 8.0 client wants to connect to a MySQL 5.7 server using `utf8mb4` as the client character set. A client that specifies `--default-character-set=utf8mb4` is able to connect to the server. However, as in the previous example, the server falls back to its default character set and collation, not what the client requested:

```
mysql> SHOW SESSION VARIABLES LIKE 'character_set_%';
+--------------------------+--------+
| Variable_name            | Value  |
|--------------------------+--------+
| character_set_client     | latin1 |
| character_set_connection | latin1 |
|                         | latin1 |
|                         | latin1 |
| character_set_results    | latin1 |
|                         | latin1 |
```

Why does this occur? After all, `utf8mb4` is known to the 8.0 client and the 5.7 server, so both of them recognize it. To understand this behavior, it is necessary to understand that when the client tells the server which character set it wants to use, it really tells the server the default collation for that character set. Therefore, the aforementioned behavior occurs due to a combination of factors:

- The default collation for `utf8mb4` differs between MySQL 5.7 and 8.0 (`utf8mb4_general_ci` for 5.7, `utf8mb4_0900_ai_ci` for 8.0).
- When the 8.0 client requests a character set of `utf8mb4`, what it sends to the server is the default 8.0 `utf8mb4` collation; that is, the `utf8mb4_0900_ai_ci`.
- `utf8mb4_0900_ai_ci` is implemented only as of MySQL 8.0, so the 5.7 server does not recognize it.
- Because the 5.7 server does not recognize `utf8mb4_0900_ai_ci`, it cannot satisfy the client character set request, and falls back to its default character set and collation (`latin1` and `latin1_swedish_ci`).

In this case, the client can still use `utf8mb4` by issuing a `SET NAMES 'utf8mb4'` statement after connecting. The resulting collation is the 5.7 default `utf8mb4` collation; that is, `utf8mb4_general_ci`. If the client additionally wants a collation of `utf8mb4_0900_ai_ci`, it cannot achieve that because the server does not recognize that collation. The client must either be willing to use a different `utf8mb4` collation, or connect to a server from MySQL 8.0 or higher.

**Runtime Error Handling**

Within an established connection, the client can request a change of connection character set and collation with `SET NAMES` or `SET CHARACTER SET`.

Some character sets cannot be used as the client character set; see **Impermissible Client Character Sets**. If you specify a character set that is valid but not permitted as a client character set, the server returns an error:

```
mysql> SET NAMES 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client' can't be set to the value of 'ucs2'
```

If the server does not recognize the character set (or the collation), it produces an error:

```
mysql> SET NAMES 'bogus';
```
Configuring Application Character Set and Collation

For applications that store data using the default MySQL character set and collation (\texttt{latin1}, \texttt{latin1\_swedish\_ci}), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might use the default of \texttt{latin1}, whereas applications that use another database might use \texttt{sjis}.

- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.

- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings as the defaults for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the \texttt{utf8} character set and \texttt{utf8\_general\_ci} collation in particular contexts as an alternative to the defaults of \texttt{latin1} and \texttt{latin1\_swedish\_ci}.

- **Specify character settings per database.** To create a database such that its tables use a given default character set and collation for data storage, use a \texttt{CREATE DATABASE} statement like this:

```
CREATE DATABASE mydb
    CHARACTER SET utf8
    COLLATE utf8\_general\_ci;
```

Tables created in the database use \texttt{utf8} and \texttt{utf8\_general\_ci} by default for any character columns.

Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a \texttt{SET NAMES 'utf8'} statement after connecting. The statement can be used regardless of connection method (the \texttt{mysql} client, PHP scripts, and so forth).

In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, to connect using \texttt{mysql}, you can specify the \texttt{--default-character-set=utf8} command-line option to achieve the same effect as \texttt{SET NAMES 'utf8'}.

For more information about configuring client connections, see Section 1.4, “Connection Character Sets and Collations”.

---

**Tip**

A client that wants to verify whether its requested character set was honored by the server can execute the following statement after connecting and checking that the result is the expected character set:

```
SELECT @@character_set_client;
```
If you use `ALTER DATABASE` to change the database default character set or

collation, existing stored routines in the database that use those defaults must

be dropped and recreated so that they use the new defaults. (In a stored routine,

variables with character data types use the database defaults if the character set

or collation are not specified explicitly. See `CREATE PROCEDURE` and `CREATE

FUNCTION Statements`.)

### Specifies character settings at server startup.

To select a character set and collation at server startup, use the `--character-set-server` and `--collation-server` options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
```

These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

It is still necessary for applications to configure their connection using `SET NAMES` or equivalent after they connect, as described previously. You might be tempted to start the server with the `--init_connect="SET NAMES 'utf8'"` option to cause `SET NAMES` to be executed automatically for each client that connects. However, this may yield inconsistent results because the `init_connect` value is not executed for users who have the `SUPER` privilege.

### Specifies character settings at MySQL configuration time.

To select a character set and collation if you configure and build MySQL from source, use the `DEFAULT_CHARSET` and `DEFAULT_COLLATION` `CMake` options:

```
cmake . -DDEFAULT_CHARSET=utf8
-DDEFAULT_COLLATION=utf8_general_ci
```

The resulting server uses `utf8` and `utf8_general_ci` as the default for databases and tables and for client connections. It is unnecessary to use `--character-set-server` and `--collation-server` to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using `SET NAMES` or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. For example, if you send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

### 1.6 Error Message Character Set

This section describes how the MySQL server uses character sets for constructing error messages. For information about the language of error messages (rather than the character set), see Section 1.12, “Setting the Error Message Language”. For general information about configuring error logging, see The Error Log.
Character Set for Error Message Construction

The server constructs error messages as follows:

- The message template uses UTF-8 (\texttt{utf8mb3}).
- Parameters in the message template are replaced with values that apply to a specific error occurrence:
  - Identifiers such as table or column names use UTF-8 internally so they are copied as is.
  - Character (nonbinary) string values are converted from their character set to UTF-8.
  - Binary string values are copied as is for bytes in the range \texttt{0x20} to \texttt{0x7E}, and using \texttt{\textbackslash{}x} hexadecimal encoding for bytes outside that range. For example, if a duplicate-key error occurs for an attempt to insert \texttt{0x41CF9F} into a \texttt{VARBINARY} unique column, the resulting error message uses UTF-8 with some bytes hexadecimal encoded:

\begin{verbatim}
Duplicate entry 'A\textbackslash{}CF\textbackslash{}9F' for key 1
\end{verbatim}

Character Set for Error Message Disposition

An error message, once constructed, can be written by the server to the error log or sent to clients:

- If the server writes the error message to the error log, it writes it in UTF-8, as constructed, without conversion to another character set.
- If the server sends the error message to a client program, the server converts it from UTF-8 to the character set specified by the \texttt{character_set_results} system variable. If \texttt{character_set_results} has a value of \texttt{NULL} or \texttt{binary}, no conversion occurs. No conversion occurs if the variable value is \texttt{utf8mb3} or \texttt{utf8mb4}, either, because those character sets have a repertoire that includes all UTF-8 characters used in message construction.

If characters cannot be represented in \texttt{character_set_results}, some encoding may occur during the conversion. The encoding uses Unicode code point values:

- Characters in the Basic Multilingual Plane (BMP) range (\texttt{0x0000} to \texttt{0xFFFF}) are written using \texttt{\textbackslash{}nnnn} notation.
- Characters outside the BMP range (\texttt{0x10000} to \texttt{0x10FFFF}) are written using \texttt{\textbackslash{}+nnnnnn} notation.

Clients can set \texttt{character_set_results} to control the character set in which they receive error messages. The variable can be set directly, or indirectly by means such as \texttt{SET NAMES}. For more information about \texttt{character_set_results}, see Section 1.4, “Connection Character Sets and Collations”.

1.7 Column Character Set Conversion

To convert a binary or nonbinary string column to use a particular character set, use \texttt{ALTER TABLE}. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (\texttt{BINARY, VARBINARY, BLOB}), all the values that it contains must be encoded using a single character set (the character set you’re converting the column to). If you use a
Column Character Set Conversion

binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.

- If the column has a nonbinary data type (CHAR, VARCHAR, TEXT), its contents should be encoded in the column character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a nonbinary column with the desired character set.

Suppose that a table t has a binary column named col1 defined as VARBINARY(50). Assuming that the information in the column is encoded using a single character set, you can convert it to a nonbinary column that has that character set. For example, if col1 contains binary data representing characters in the greek character set, you can convert it as follows:

```sql
ALTER TABLE t MODIFY col1 VARCHAR(50) CHARACTER SET greek;
```

If your original column has a type of BINARY(50), you could convert it to CHAR(50), but the resulting values are padded with 0x00 bytes at the end, which may be undesirable. To remove these bytes, use the TRIM() function:

```sql
UPDATE t SET col1 = TRIM(TRAILING 0x00 FROM col1);
```

Suppose that table t has a nonbinary column named col1 defined as CHAR(50) CHARACTER SET latin1 but you want to convert it to use utf8 so that you can store values from many languages. The following statement accomplishes this:

```sql
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from before MySQL 4.1 where a nonbinary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored sjis values in a column, even though MySQL's default character set was different. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was latin1 and col1 is defined as CHAR(50) but its contents are sjis values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```sql
ALTER TABLE t MODIFY col1 BLOB;
```

The next step is to convert the column to a nonbinary data type with the proper character set:

```sql
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as INSERT or UPDATE after an upgrade to MySQL 4.1 or higher. In that case, MySQL would store new values in the column using latin1; the column contains a mix of sjis and latin1 values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with ALTER TABLE. For example, if you specified NOT NULL and an explicit DEFAULT value, you should also provide them in the ALTER TABLE statement. Otherwise, the resulting column definition does not include those attributes.

To convert all character columns in a table, the ALTER TABLE ... CONVERT TO CHARACTER SET charset statement may be useful. See ALTER TABLE Statement.
1.8 Collation Issues

The following sections discuss various aspects of character set collations.

1.8.1 Using COLLATE in SQL Statements

With the COLLATE clause, you can override whatever the default collation is for a comparison. COLLATE may be used in various parts of SQL statements. Here are some examples:

- **With ORDER BY:**

  ```sql
  SELECT k
  FROM t1
  ORDER BY k COLLATE latin1_german2_ci;
  ```

- **With AS:**

  ```sql
  SELECT k COLLATE latin1_german2_ci AS k1
  FROM t1;
  ```

- **With GROUP BY:**

  ```sql
  SELECT k
  FROM t1
  GROUP BY k COLLATE latin1_german2_ci;
  ```

- **With aggregate functions:**

  ```sql
  SELECT MAX(k COLLATE latin1_german2_ci)
  FROM t1;
  ```

- **With DISTINCT:**

  ```sql
  SELECT DISTINCT k COLLATE latin1_german2_ci
  FROM t1;
  ```

- **With WHERE:**

  ```sql
  SELECT *
  FROM t1
  WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
  ```

  ```sql
  SELECT *
  FROM t1
  WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
  ```

- **With HAVING:**

  ```sql
  SELECT k
  FROM t1
  GROUP BY k
  HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
  ```

1.8.2 COLLATE Clause Precedence

The COLLATE clause has high precedence (higher than ||), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```
1.8.3 Character Set and Collation Compatibility

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the latin2_bin collation is not legal with the latin1 character set:

```sql
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

1.8.4 Collation Coercibility in Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column `x`:

```sql
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example, this statement performs a comparison between the column `x` and the string literal `'Y'`:

```sql
SELECT x FROM T WHERE x = 'Y';
```

If `x` and `'Y'` have the same collation, there is no ambiguity about the collation to use for the comparison. But if they have different collations, should the comparison use the collation of `x`, or of `'Y'`? Both `x` and `'Y'` have collations, so which collation takes precedence?

A mix of collations may also occur in contexts other than comparison. For example, a multiple-argument concatenation operation such as `CONCAT(x, 'Y')` combines its arguments to produce a single string. What collation should the result have?

To resolve questions like these, MySQL checks whether the collation of one item can be coerced to the collation of the other. MySQL assigns coercibility values as follows:

- An explicit `COLLATE` clause has a coercibility of 0 (not coercible at all).
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` or `VERSION()`) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.
- The collation of a numeric or temporal value has a coercibility of 5.
- `NULL` or an expression that is derived from `NULL` has a coercibility of 6.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:
  - If both sides are Unicode, or both sides are not Unicode, it is an error.
The binary Collation Compared to _bin Collations

- If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

```
SELECT CONCAT(utf8_column, latin1_column) FROM t1;
```

It returns a result that has a character set of utf8 and the same collation as utf8_column. Values of latin1_column are automatically converted to utf8 before concatenating.

- For an operation with operands from the same character set but that mix a _bin collation and a _ci or _cs collation, the _bin collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, applied to collations rather than data types.

Although automatic conversion is not in the SQL standard, the standard does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. More generally, MySQL uses the concept of character set repertoire, which can sometimes be used to determine subset relationships among character sets and enable conversion of operands in operations that would otherwise produce an error. See Section 1.2.1, “Character Set Repertoire”.

The following table illustrates some applications of the preceding rules.

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Collation Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>column1 = 'A'</td>
<td>Use collation of column1</td>
</tr>
<tr>
<td>column1 = 'A' COLLATE x</td>
<td>Use collation of 'A' COLLATE x</td>
</tr>
<tr>
<td>column1 COLLATE x = 'A' COLLATE y</td>
<td>Error</td>
</tr>
</tbody>
</table>

To determine the coercibility of a string expression, use the `COERCIBILITY()` function (see Information Functions):

```
mysql> SELECT COERCIBILITY(_utf8'A' COLLATE utf8_bin);
-> 0
mysql> SELECT COERCIBILITY(VERSION());
-> 3
mysql> SELECT COERCIBILITY('A');
-> 4
mysql> SELECT COERCIBILITY(1000);
-> 5
mysql> SELECT COERCIBILITY(NULL);
-> 6
```

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument 1 in the expression `CONCAT(1, 'abc')`, the result is a character (nonbinary) string that has a character set and collation determined by the `character_set_connection` and `collation_connection` system variables. See Type Conversion in Expression Evaluation.

### 1.8.5 The binary Collation Compared to _bin Collations

This section describes how the binary collation for binary strings compares to _bin collations for nonbinary strings.

Binary strings (as stored using the `BINARY`, `VARBINARY`, and `BLOB` data types) have a character set and collation named binary. Binary strings are sequences of bytes and the numeric values of those bytes determine comparison and sort order. See Section 1.10.8, “The Binary Character Set”.
The binary Collation Compared to _bin Collations

Nonbinary strings (as stored using the CHAR, VARCHAR, and TEXT data types) have a character set and collation other than binary. A given nonbinary character set can have several collations, each of which defines a particular comparison and sort order for the characters in the set. One of these is the binary collation, indicated by a _bin suffix in the collation name. For example, the binary collation for utf8 and latin1 is named utf8_bin and latin1_bin, respectively.

The binary collation differs from _bin collations in several respects, discussed in the following sections:

- The Unit for Comparison and Sorting
- Character Set Conversion
- Lettercase Conversion
- Trailing Space Handling in Comparisons
- Trailing Space Handling for Inserts and Retrievals

The Unit for Comparison and Sorting

Binary strings are sequences of bytes. For the binary collation, comparison and sorting are based on numeric byte values. Nonbinary strings are sequences of characters, which might be multibyte. Collations for nonbinary strings define an ordering of the character values for comparison and sorting. For _bin collations, this ordering is based on numeric character code values, which is similar to ordering for binary strings except that character code values might be multibyte.

Character Set Conversion

A nonbinary string has a character set and is automatically converted to another character set in many cases, even when the string has a _bin collation:

- When assigning column values to another column that has a different character set:

  ```sql
  UPDATE t1 SET utf8_bin_column=latin1_column;
  INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
  ```

- When assigning column values for INSERT or UPDATE using a string literal:

  ```sql
  SET NAMES latin1;
  INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
  ```

- When sending results from the server to a client:

  ```sql
  SET NAMES latin1;
  SELECT utf8_bin_column FROM t2;
  ```

For binary string columns, no conversion occurs. For cases similar to those preceding, the string value is copied byte-wise.

Lettercase Conversion

Collations for nonbinary character sets provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for _bin collations that ignore lettercase for ordering:

```sql
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_bin;
mysql> SELECT LOWER('aA'), UPPER('zZ');
+-------------+-------------+
<table>
<thead>
<tr>
<th>LOWER('aA')</th>
<th>UPPER('zZ')</th>
</tr>
</thead>
<tbody>
<tr>
<td>aa</td>
<td>ZZ</td>
</tr>
</tbody>
</table>
```
Examples of the Effect of Collation

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must first be converted to a nonbinary string using a character set appropriate for the data stored in the string:

```
mysql> SET NAMES binary;
mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING utf8mb4));
+-------------+------------------------------------+
| LOWER('aA') | LOWER(CONVERT('aA' USING utf8mb4)) |
| aA          | aa                                 |
+-------------+------------------------------------+
```

Trailing Space Handling in Comparisons

Nonbinary strings have PAD SPACE behavior for all collations, including _bin collations. Trailing spaces are insignificant in comparisons:

```
mysql> SET NAMES utf8 COLLATE utf8_bin;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          1 |
+------------+
```

For binary strings, all bytes are significant in comparisons, including trailing spaces:

```
mysql> SET NAMES binary;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
```

Trailing Space Handling for Inserts and Retrievals

CHAR(\(N\)) columns store nonbinary strings \(N\) characters long. For inserts, values shorter than \(N\) characters are extended with spaces. For retrievals, trailing spaces are removed.

BINARY(\(N\)) columns store binary strings \(N\) bytes long. For inserts, values shorter than \(N\) bytes are extended with 0x00 bytes. For retrievals, nothing is removed; a value of the declared length is always returned.

```
mysql> CREATE TABLE t1 (
    a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
    b BINARY(10)
);
mysql> INSERT INTO t1 VALUES ('x','x');
mysql> INSERT INTO t1 VALUES ('x ','x ');
mysql> SELECT a, b, HEX(a), HEX(b) FROM t1;
+------+------------+--------+----------------------+
| a    | b          | HEX(a) | HEX(b)               |
| x    | x          | 78     | 78000000000000000000 |
| x    | x          | 78     | 78200000000000000000 |
+------+------------+--------+----------------------+
```

1.8.6 Examples of the Effect of Collation

Example 1: Sorting German Umlauts
Examples of the Effect of Collation

Suppose that column \( X \) in table \( T \) has these \texttt{latin1} column values:

- Muffler
- Müller
- MX Systems
- MySQL

Suppose also that the column values are retrieved using the following statement:

\[
\text{SELECT } X \text{ FROM } T \text{ ORDER BY } X \text{ COLLATE } \text{collation\_name};
\]

The following table shows the resulting order of the values if we use \texttt{ORDER BY} with different collations.

<table>
<thead>
<tr>
<th>\texttt{latin1_swedish_ci}</th>
<th>\texttt{latin1_german1_ci}</th>
<th>\texttt{latin1_german2_ci}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muffler</td>
<td>Muffler</td>
<td>Müller</td>
</tr>
<tr>
<td>MX Systems</td>
<td>Müller</td>
<td>Muffler</td>
</tr>
<tr>
<td>Müller</td>
<td>MX Systems</td>
<td>MX Systems</td>
</tr>
<tr>
<td>MySQL</td>
<td>MySQL</td>
<td>MySQL</td>
</tr>
</tbody>
</table>

The character that causes the different sort orders in this example is the U with two dots over it (\( \ddot{u} \)), which the Germans call “U-umlaut.”

- The first column shows the result of the \texttt{SELECT} using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the \texttt{SELECT} using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the \texttt{SELECT} using the German DIN-2 rule, which says that U-umlaut sorts with UE.

**Example 2: Searching for German Umlauts**

Suppose that you have three tables that differ only by the character set and collation used:

```sql
mysql> \texttt{SET NAMES utf8;}
mysql> \texttt{CREATE TABLE german1 (}
   \texttt{  \texttt{c }\texttt{ CHAR(10)}}
   \texttt{  \texttt{) CHARACTER SET latin1 COLLATE latin1\_german1\_ci;}}
mysql> \texttt{CREATE TABLE german2 (}
   \texttt{  \texttt{c }\texttt{ CHAR(10)}}
   \texttt{  \texttt{) CHARACTER SET latin1 COLLATE latin1\_german2\_ci;}}
mysql> \texttt{CREATE TABLE germanutf8 (}
   \texttt{  \texttt{c }\texttt{ CHAR(10)}}
   \texttt{  \texttt{) CHARACTER SET utf8 COLLATE utf8\_unicode\_ci;}}
```

Each table contains two records:

```sql
mysql> \texttt{INSERT INTO german1 VALUES ('Bar'), ('Bär');}
mysql> \texttt{INSERT INTO german2 VALUES ('Bar'), ('Bär');}
mysql> \texttt{INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');}
```

Two of the above collations have an \( \texttt{A = Ä} \) equality, and one has no such equality (\texttt{latin1\_german2\_ci}). For that reason, you’ll get these results in comparisons:

```sql
mysql> \texttt{SELECT * FROM german1 WHERE c = 'Bär';}
<table>
<thead>
<tr>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar</td>
</tr>
<tr>
<td>Bär</td>
</tr>
</tbody>
</table>
```
This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

1.8.7 Using Collation in INFORMATION_SCHEMA Searches

String columns in `INFORMATION_SCHEMA` tables have a collation of `utf8_general_ci`, which is case-insensitive. However, for values that correspond to objects that are represented in the file system, such as databases and tables, searches in `INFORMATION_SCHEMA` string columns can be case-sensitive or case-insensitive, depending on the characteristics of the underlying file system and the value of the `lower_case_table_names` system variable. For example, searches may be case-sensitive if the file system is case-sensitive. This section describes this behavior and how to modify it if necessary; see also Bug #34921.

Suppose that a query searches the `SCHEMATA.SCHEMA_NAME` column for the `test` database. On Linux, file systems are case-sensitive, so comparisons of `SCHEMATA.SCHEMA_NAME` with `'test'` match, but comparisons with `'TEST'` do not:

```sql
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    WHERE SCHEMA_NAME = 'test';
```
```
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
```

These results occur with the `lower_case_table_names` system variable set to 0. Changing the value of `lower_case_table_names` to 1 or 2 causes the second query to return the same (nonempty) result as the first query.

On Windows or macOS, file systems are not case-sensitive, so comparisons match both `'test'` and `'TEST'`:

```sql
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    WHERE SCHEMA_NAME = 'test';
```
```
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
```

```sql
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
    WHERE SCHEMA_NAME = 'TEST';
```
```
+-------------+
| SCHEMA_NAME |
+-------------+
```
```
The value of `lower_case_table_names` makes no difference in this context.

The preceding behavior occurs because the `utf8_general_ci` collation is not used for `INFORMATION_SCHEMA` queries when searching for values that correspond to objects represented in the file system. It is a result of file system-scanning optimizations implemented for `INFORMATION_SCHEMA` searches. For information about these optimizations, see Optimizing `INFORMATION_SCHEMA` Queries.

If the result of a string operation on an `INFORMATION_SCHEMA` column differs from expectations, a workaround is to use an explicit `COLLATE` clause to force a suitable collation (see Section 1.8.1, “Using `COLLATE` in SQL Statements”). For example, to perform a case-insensitive search, use `COLLATE` with the `INFORMATION_SCHEMA` column name:

```sql
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
mysql>
```

In the preceding queries, it is important to apply the `COLLATE` clause to the `INFORMATION_SCHEMA` column name. Applying `COLLATE` to the comparison value has no effect.

You can also use the `UPPER()` or `LOWER()` function:

```sql
WHERE UPPER(SCHEMA_NAME) = 'TEST'
WHERE LOWER(SCHEMA_NAME) = 'test'
```

Although a case-insensitive comparison can be performed even on platforms with case-sensitive file systems, as just shown, it is not necessarily always the right thing to do. On such platforms, it is possible to have multiple objects with names that differ only in lettercase. For example, tables named `city`, `CITY`, and `City` can all exist simultaneously. Consider whether a search should match all such names or just one and write queries accordingly. The first of the following comparisons (with `utf8_bin`) is case-sensitive; the others are not:

```sql
WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'
```

Searches in `INFORMATION_SCHEMA` string columns for values that refer to `INFORMATION_SCHEMA` itself do use the `utf8_general_ci` collation because `INFORMATION_SCHEMA` is a “virtual” database not represented in the file system. For example, comparisons with `SCHEMATA.SCHEMA_NAME` match `'information_schema'` or `'INFORMATION_SCHEMA'` regardless of platform:

```sql
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
WHERE SCHEMA_NAME = 'information_schema';
+--------------------+
| SCHEMA_NAME         |
+--------------------+
| information_schema  |
+--------------------+
```
1.9 Unicode Support

The Unicode Standard includes characters from the Basic Multilingual Plane (BMP) and supplementary characters that lie outside the BMP. This section describes support for Unicode in MySQL. For information about the Unicode Standard itself, visit the Unicode Consortium website.

BMP characters have these characteristics:

- Their code point values are between 0 and 65535 (or \texttt{U+0000} and \texttt{U+FFFF}).
- They can be encoded in a variable-length encoding using 8, 16, or 24 bits (1 to 3 bytes).
- They can be encoded in a fixed-length encoding using 16 bits (2 bytes).
- They are sufficient for almost all characters in major languages.

Supplementary characters lie outside the BMP:

- Their code point values are between \texttt{U+10000} and \texttt{U+10FFFF}).
- Unicode support for supplementary characters requires character sets that have a range outside BMP characters and therefore take more space than BMP characters (up to 4 bytes per character).

The UTF-8 (Unicode Transformation Format with 8-bit units) method for encoding Unicode data is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

MySQL supports these Unicode character sets:

- \texttt{utf8mb4}: A UTF-8 encoding of the Unicode character set using one to four bytes per character.
- \texttt{utf8mb3}: A UTF-8 encoding of the Unicode character set using one to three bytes per character.
- \texttt{utf8}: An alias for \texttt{utf8mb3}.
- \texttt{ucs2}: The UCS-2 encoding of the Unicode character set using two bytes per character.
- \texttt{utf16}: The UTF-16 encoding for the Unicode character set using two or four bytes per character. Like \texttt{ucs2} but with an extension for supplementary characters.
- \texttt{utf16le}: The UTF-16LE encoding for the Unicode character set. Like \texttt{utf16} but little-endian rather than big-endian.
- \texttt{utf32}: The UTF-32 encoding for the Unicode character set using four bytes per character.
The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)

Table 1.2, “Unicode Character Set General Characteristics”, summarizes the general characteristics of Unicode character sets supported by MySQL.

Table 1.2 Unicode Character Set General Characteristics

<table>
<thead>
<tr>
<th>Character Set</th>
<th>Supported Characters</th>
<th>Required Storage Per Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>utf8mb3, utf8</td>
<td>BMP only</td>
<td>1, 2, or 3 bytes</td>
</tr>
<tr>
<td>ucs2</td>
<td>BMP only</td>
<td>2 bytes</td>
</tr>
<tr>
<td>utf8mb4</td>
<td>BMP and supplementary</td>
<td>1, 2, 3, or 4 bytes</td>
</tr>
<tr>
<td>utf16</td>
<td>BMP and supplementary</td>
<td>2 or 4 bytes</td>
</tr>
<tr>
<td>utf16le</td>
<td>BMP and supplementary</td>
<td>2 or 4 bytes</td>
</tr>
<tr>
<td>utf32</td>
<td>BMP and supplementary</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

Characters outside the BMP compare as REPLACEMENT CHARACTER and convert to '¿' when converted to a Unicode character set that supports only BMP characters (utf8mb3 or ucs2).

If you use character sets that support supplementary characters and thus are “wider” than the BMP-only utf8mb3 and ucs2 character sets, there are potential incompatibility issues for your applications; see Section 1.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”. That section also describes how to convert tables from the (3-byte) utf8mb3 to the (4-byte) utf8mb4, and what constraints may apply in doing so.

A similar set of collations is available for most Unicode character sets. For example, each has a Danish collation, the names of which are utf8mb4_danish_ci, utf8mb3_danish_ci, utf8_danish_ci, ucs2_danish_ci, utf16_danish_ci, and utf32_danish_ci. The exception is utf16le, which has only two collations. For information about Unicode collations and their differentiating properties, including collation properties for supplementary characters, see Section 1.10.1, “Unicode Character Sets”.

The MySQL implementation of UCS-2, UTF-16, and UTF-32 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of values must be performed when transferring data between those systems and MySQL. The implementation of UTF-16LE is little-endian.

MySQL uses no BOM for UTF-8 values.

Client applications that communicate with the server using Unicode should set the client character set accordingly (for example, by issuing a SET NAMES 'utf8mb4' statement). Some character sets cannot be used as the client character set. Attempting to use them with SET NAMES or SET CHARACTER SET produces an error. See Impermissible Client Character Sets.

The following sections provide additional detail on the Unicode character sets in MySQL.

1.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)

The utfmb4 character set has these characteristics:

• Supports BMP and supplementary characters.

• Requires a maximum of four bytes per multibyte character.

utf8mb4 contrasts with the utf8mb3 character set, which supports only BMP characters and uses a maximum of three bytes per character:

• For a BMP character, utf8mb4 and utf8mb3 have identical storage characteristics: same code values, same encoding, same length.
• For a supplementary character, utf8mb4 requires four bytes to store it, whereas utf8mb3 cannot store the character at all. When converting utf8mb3 columns to utf8mb4, you need not worry about converting supplementary characters because there are none.

utf8mb4 is a superset of utf8mb3, so for an operation such as the following concatenation, the result has character set utf8mb4 and the collation of utf8mb4_coll:

```sql
SELECT CONCAT(utf8mb3_col, utf8mb4_col);
```

Similarly, the following comparison in the WHERE clause works according to the collation of utf8mb4_coll:

```sql
SELECT * FROM utf8mb3_tbl, utf8mb4_tbl
WHERE utf8mb3_tbl.utf8mb3_col = utf8mb4_tbl.utf8mb4_col;
```

For information about data type storage as it relates to multibyte character sets, see String Type Storage Requirements.

### 1.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding)

The utf8mb3 character set has these characteristics:

• Supports BMP characters only (no support for supplementary characters)
• Requires a maximum of three bytes per multibyte character.

Applications that use UTF-8 data but require supplementary character support should use utf8mb4 rather than utf8mb3 (see Section 1.9.1, “The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)”).

Exactly the same set of characters is available in utf8mb3 and ucs2. That is, they have the same repertoire.

utf8 is an alias for utf8mb3; the character limit is implicit, rather than explicit in the name.

utf8mb3 can be used in CHARACTER SET clauses, and utf8mb3_collation_substring in COLLATE clauses, where collation_substring is bin, czech_ci, danish_ci, esperanto_ci, estonian_ci, and so forth. For example:

```sql
CREATE TABLE t (s1 CHAR(1) CHARACTER SET utf8mb3;
SELECT * FROM t WHERE s1 COLLATE utf8mb3_general_ci = 'x';
DECLARE x VARCHAR(5) CHARACTER SET utf8mb3 COLLATE utf8mb3_danish_ci;
SELECT CAST('a' AS CHAR CHARACTER SET utf8) COLLATE utf8_czech_ci;
```

MySQL immediately converts instances of utf8mb3 in statements to utf8, so in statements such as SHOW CREATE TABLE or SELECT CHARACTER_SET_NAME FROM INFORMATION_SCHEMA.COLUMNS or SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLUMNS, users see the name utf8 or utf8_collation_substring.

utf8mb3 is also valid in contexts other than CHARACTER SET clauses. For example:

```sql
mysqld --character-set-server=utf8mb3
```

set NAMES 'utf8mb3'; /* and other SET statements that have similar effect */
SELECT '_utf8mb3' 'a';

For information about data type storage as it relates to multibyte character sets, see String Type Storage Requirements.

### 1.9.3 The utf8 Character Set (Alias for utf8mb3)

utf8 is an alias for the utf8mb3 character set. For more information, see Section 1.9.2, “The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding)”. 
1.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a 2-byte Unicode code with the most significant byte first. For example: LATIN CAPITAL LETTER A has the code 0x0041 and it is stored as a 2-byte sequence: 0x00 0x41. CYRILLIC SMALL LETTER YERU (Unicode 0x044B) is stored as a 2-byte sequence: 0x04 0x4B. For Unicode characters and their codes, please refer to the Unicode Consortium website.

The ucs2 character set has these characteristics:

- Supports BMP characters only (no support for supplementary characters)
- Uses a fixed-length 16-bit encoding and requires two bytes per character.

1.9.5 The utf16 Character Set (UTF-16 Unicode Encoding)

The utf16 character set is the ucs2 character set with an extension that enables encoding of supplementary characters:

- For a BMP character, utf16 and ucs2 have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, utf16 has a special sequence for representing the character using 32 bits. This is called the "surrogate" mechanism: For a number greater than 0xffff, take 10 bits and add them to 0xd800 and put them in the first 16-bit word, take 10 more bits and add them to 0xdc00 and put them in the next 16-bit word. Consequently, all supplementary characters require 32 bits, where the first 16 bits are a number between 0xd800 and 0xdbff, and the last 16 bits are a number between 0xdc00 and 0xdfff. Examples are in Section 15.5 Surrogates Area of the Unicode 4.0 document.

Because utf16 supports surrogates and ucs2 does not, there is a validity check that applies only in utf16: You cannot insert a top surrogate without a bottom surrogate, or vice versa. For example:

```sql
INSERT INTO t (ucs2_column) VALUES (0xd800); /* legal */
INSERT INTO t (utf16_column) VALUES (0xd800); /* illegal */
```

There is no validity check for characters that are technically valid but are not true Unicode (that is, characters that Unicode considers to be "unassigned code points" or "private use" characters or even "illegals" like 0xffff). For example, since U+F8FF is the Apple Logo, this is legal:

```sql
INSERT INTO t (utf16_column) VALUES (0xf8ff); /* legal */
```

Such characters cannot be expected to mean the same thing to everyone.

Because MySQL must allow for the worst case (that one character requires four bytes) the maximum length of a utf16 column or index is only half of the maximum length for a ucs2 column or index. For example, the maximum length of a MEMORY table index key is 3072 bytes, so these statements create tables with the longest permitted indexes for ucs2 and utf16 columns:

```sql
CREATE TABLE tf (s1 VARCHAR(1536) CHARACTER SET ucs2) ENGINE=MEMORY;
CREATE INDEX i ON tf (s1);
CREATE TABLE tg (s1 VARCHAR(768) CHARACTER SET utf16) ENGINE=MEMORY;
CREATE INDEX i ON tg (s1);
```

1.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding)

This is the same as utf16 but is little-endian rather than big-endian.

1.9.7 The utf32 Character Set (UTF-32 UnicodeEncoding)
Converting Between 3-Byte and 4-Byte Unicode Character Sets

The utf32 character set is fixed length (like ucs2 and unlike utf16). utf32 uses 32 bits for every character, unlike ucs2 (which uses 16 bits for every character), and unlike utf16 (which uses 16 bits for some characters and 32 bits for others).

utf32 takes twice as much space as ucs2 and more space than utf16, but utf32 has the same advantage as ucs2 that it is predictable for storage: The required number of bytes for utf32 equals the number of characters times 4. Also, unlike utf16, there are no tricks for encoding in utf32, so the stored value equals the code value.

To demonstrate how the latter advantage is useful, here is an example that shows how to determine a utf8mb4 value given the utf32 code value:

```sql
/* Assume code value = 100cc LINEAR B WHEELED CHARIOT */
CREATE TABLE tmp (utf32_col CHAR(1) CHARACTER SET utf32,
                   utf8mb4_col CHAR(1) CHARACTER SET utf8mb4);
INSERT INTO tmp VALUES (0x000100cc,NULL);
UPDATE tmp SET utf8mb4_col = utf32_col;
SELECT HEX(utf32_col),HEX(utf8mb4_col) FROM tmp;
```

MySQL is very forgiving about additions of unassigned Unicode characters or private-use-area characters. There is in fact only one validity check for utf32: No code value may be greater than 0x10ffff. For example, this is illegal:

```sql
INSERT INTO t (utf32_column) VALUES (0x110000); /* illegal */
```

1.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets

This section describes issues that you may face when converting character data between the utf8mb3 and utf8mb4 character sets.

Note

This discussion focuses primarily on converting between utf8mb3 and utf8mb4, but similar principles apply to converting between the ucs2 character set and character sets such as utf16 or utf32.

The utf8mb3 and utf8mb4 character sets differ as follows:

- utf8mb3 supports only characters in the Basic Multilingual Plane (BMP). utf8mb4 additionally supports supplementary characters that lie outside the BMP.
- utf8mb3 uses a maximum of three bytes per character. utf8mb4 uses a maximum of four bytes per character.

Note

This discussion refers to the utf8mb3 and utf8mb4 character set names to be explicit about referring to 3-byte and 4-byte UTF-8 character set data. The exception is that in table definitions, utf8 is used because MySQL converts instances of utf8mb3 specified in such definitions to utf8, which is an alias for utf8mb3.

One advantage of converting from utf8mb3 to utf8mb4 is that this enables applications to use supplementary characters. One tradeoff is that this may increase data storage space requirements.

In terms of table content, conversion from utf8mb3 to utf8mb4 presents no problems:

- For a BMP character, utf8mb4 and utf8mb3 have identical storage characteristics: same code values, same encoding, same length.
• For a supplementary character, utf8mb4 requires four bytes to store it, whereas utf8mb3 cannot store the character at all. When converting utf8mb3 columns to utf8mb4, you need not worry about converting supplementary characters because there are none.

In terms of table structure, these are the primary potential incompatibilities:

• For the variable-length character data types (VARCHAR and the TEXT types), the maximum permitted length in characters is less for utf8mb4 columns than for utf8mb3 columns.

• For all character data types (CHAR, VARCHAR, and the TEXT types), the maximum number of characters that can be indexed is less for utf8mb4 columns than for utf8mb3 columns.

Consequently, to convert tables from utf8mb3 to utf8mb4, it may be necessary to change some column or index definitions.

Tables can be converted from utf8mb3 to utf8mb4 by using ALTER TABLE. Suppose that a table has this definition:

    CREATE TABLE t1 (  
    col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,  
    col2 CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL  
) CHARACTER SET utf8;

The following statement converts t1 to use utf8mb4:

    ALTER TABLE t1  
    DEFAULT CHARACTER SET utf8mb4,  
    MODIFY col1 CHAR(10)  
    CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,  
    MODIFY col2 CHAR(10)  
    CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL;

The catch when converting from utf8mb3 to utf8mb4 is that the maximum length of a column or index key is unchanged in terms of bytes. Therefore, it is smaller in terms of characters because the maximum length of a character is four bytes instead of three. For the CHAR, VARCHAR, and TEXT data types, watch for these issues when converting your MySQL tables:

• Check all definitions of utf8mb3 columns and make sure they do not exceed the maximum length for the storage engine.

• Check all indexes on utf8mb3 columns and make sure they do not exceed the maximum length for the storage engine. Sometimes the maximum can change due to storage engine enhancements.

If the preceding conditions apply, you must either reduce the defined length of columns or indexes, or continue to use utf8mb3 rather than utf8mb4.

Here are some examples where structural changes may be needed:

• A TINYTEXT column can hold up to 255 bytes, so it can hold up to 85 3-byte or 63 4-byte characters. Suppose that you have a TINYTEXT column that uses utf8mb3 but must be able to contain more than 63 characters. You cannot convert it to utf8mb4 unless you also change the data type to a longer type such as TEXT.

    Similarly, a very long VARCHAR column may need to be changed to one of the longer TEXT types if you want to convert it from utf8mb3 to utf8mb4.

• InnoDB has a maximum index length of 767 bytes for tables that use COMPACT or REDUNDANT row format, so for utf8mb3 or utf8mb4 columns, you can index a maximum of 255 or 191 characters, respectively. If you currently have utf8mb3 columns with indexes longer than 191 characters, you must index a smaller number of characters.
In an InnoDB table that uses COMPACT or REDUNDANT row format, these column and index definitions are legal:

```sql
col1 VARCHAR(500) CHARACTER SET utf8, INDEX (col1(255))
```

To use utf8mb4 instead, the index must be smaller:

```sql
col1 VARCHAR(500) CHARACTER SET utf8mb4, INDEX (col1(191))
```

**Note**

For InnoDB tables that use COMPRESSED or DYNAMIC row format, you can enable the innodb_large_prefix option to permit index key prefixes longer than 767 bytes (up to 3072 bytes). Creating such tables also requires the option values innodb_file_format=barracuda and innodb_file_per_table=true. In this case, enabling the innodb_large_prefix option enables you to index a maximum of 1024 or 768 characters for utf8mb3 or utf8mb4 columns, respectively. For related information, see InnoDB Limits.

The preceding types of changes are most likely to be required only if you have very long columns or indexes. Otherwise, you should be able to convert your tables from utf8mb3 to utf8mb4 without problems, using `ALTER TABLE` as described previously.

The following items summarize other potential incompatibilities:

- **SET NAMES 'utf8mb4'** causes use of the 4-byte character set for connection character sets. As long as no 4-byte characters are sent from the server, there should be no problems. Otherwise, applications that expect to receive a maximum of three bytes per character may have problems. Conversely, applications that expect to send 4-byte characters must ensure that the server understands them.

- For replication, if character sets that support supplementary characters are to be used on the source, all replicas must understand them as well.

Also, keep in mind the general principle that if a table has different definitions on the source and replica, this can lead to unexpected results. For example, the differences in maximum index key length make it risky to use utf8mb3 on the source and utf8mb4 on the replica.

If you have converted to utf8mb4, utf16, utf16le, or utf32, and then decide to convert back to utf8mb3 or ucs2 (for example, to downgrade to an older version of MySQL), these considerations apply:

- utf8mb3 and ucs2 data should present no problems.

- The server must be recent enough to recognize definitions referring to the character set from which you are converting.

- For object definitions that refer to the utf8mb4 character set, you can dump them with `mysqldump` prior to downgrading, edit the dump file to change instances of utf8mb4 to utf8, and reload the file in the older server, as long as there are no 4-byte characters in the data. The older server sees utf8 in the dump file object definitions and creates new objects that use the (3-byte) utf8 character set.

## 1.10 Supported Character Sets and Collations

This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.
To list the available character sets and their default collations, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table. For example:

```sql
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                  | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| armSCII8 | ARMSCII-8 Armenian          | armSCII8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
| macce    | Mac Central European        | macce_general_ci    |      1 |
| macroman | Mac West European           | macroman_general_ci |      1 |
| cp852    | DOS Central European        | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |      1 |
| utf8mb4  | UTF-8 Unicode               | utf8mb4_general_ci  |      4 |
| cpl251   | Windows Cyrillic            | cpl251_general_ci   |      1 |
| utf16    | UTF-16 Unicode              | utf16_general_ci    |      4 |
| utf16le  | UTF-16LE Unicode            | utf16le_general_ci  |      4 |
| cpl256   | Windows Arabic              | cpl256_general_ci   |      1 |
| cpl257   | Windows Baltic              | cpl257_general_ci   |      1 |
| utf32    | UTF-32 Unicode              | utf32_general_ci    |      4 |
| binary   | Binary pseudo charset       | binary              |      1 |
| geostd8  | GEOSTD8 Georgian             | geostd8_general_ci  |      1 |
| cp932    | SJSIS for Windows Japanese  | cp932_japanese_ci   |      2 |
| eucjpm8s| UJIS for Windows Japanese   | eucjpm8s_japanese_ci|      2 |
+----------+-----------------------------+---------------------+--------+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

### 1.10.1 Unicode Character Sets

This section describes the collations available for Unicode character sets and their differentiating properties. For general information about Unicode, see Section 1.9, “Unicode Support”.

MySQL supports multiple Unicode character sets:

- **utf8mb4**: A UTF-8 encoding of the Unicode character set using one to four bytes per character.
- **utf8mb3**: A UTF-8 encoding of the Unicode character set using one to three bytes per character.
Unicode Character Sets

- **utf8**: An alias for **utf8mb3**.
- **ucs2**: The UCS-2 encoding of the Unicode character set using two bytes per character.
- **utf16**: The UTF-16 encoding for the Unicode character set using two or four bytes per character. Like **ucs2** but with an extension for supplementary characters.
- **utf16le**: The UTF-16LE encoding for the Unicode character set. Like **utf16** but little-endian rather than big-endian.
- **utf32**: The UTF-32 encoding for the Unicode character set using four bytes per character.

**utf8mb4**, **utf16**, **utf16le**, and **utf32** support Basic Multilingual Plane (BMP) characters and supplementary characters that lie outside the BMP. **utf8** and **ucs2** support only BMP characters.

Most Unicode character sets have a general collation (indicated by `_general` in the name or by the absence of a language specifier), a binary collation (indicated by `_bin` in the name), and several language-specific collations (indicated by language specifiers). For example, for **utf8mb4**, **utf8mb4_general_ci** and **utf8mb4_bin** are its general and binary collations, and **utf8mb4_danish_ci** is one of its language-specific collations.

Collation support for **utf16le** is limited. The only collations available are **utf16le_general_ci** and **utf16le_bin**. These are similar to **utf16_general_ci** and **utf16_bin**.

- **Unicode Collation Algorithm (UCA) Versions**
- **Language-Specific Collations**
- **_general_ci Versus_ _unicode_ci Collations**
- **Character Collating Weights**
- **Miscellaneous Information**

### Unicode Collation Algorithm (UCA) Versions

MySQL implements the xxx_unicode_ci collations according to the Unicode Collation Algorithm (UCA) described at [http://www.unicode.org/reports/tr10/](http://www.unicode.org/reports/tr10/). The collation uses the version-4.0.0 UCA weight keys: [http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt](http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt). The xxx_unicode_ci collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported, and combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo. A combined character is considered different from the same character written with a single unicode character in string comparisons, and the two characters are considered to have a different length (for example, as returned by the `CHAR_LENGTH()` function or in result set metadata).

Unicode collations based on UCA versions higher than 4.0.0 include the version in the collation name. Thus, **utf8_unicode_520_ci** is based on UCA 5.2.0 weight keys ([http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt](http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt)).

The LOWER() and UPPER() functions perform case folding according to the collation of their argument. A character that has uppercase and lowercase versions only in a Unicode version higher than 4.0.0 is converted by these functions only if the argument collation uses a high enough UCA version.

### Language-Specific Collations

MySQL implements language-specific Unicode collations if the ordering based only on the Unicode Collation Algorithm (UCA) does not work well for a language. Language-specific collations are UCA-based,
with additional language tailoring rules. Examples of such rules appear later in this section. For questions about particular language orderings, unicode.org provides Common Locale Data Repository (CLDR) collation charts at http://www.unicode.org/cldr/charts/30/collation/index.html.

A language name shown in the following table indicates a language-specific collation. Unicode character sets may include collations for one or more of these languages.

### Table 1.3 Unicode Collation Language Specifiers

<table>
<thead>
<tr>
<th>Language</th>
<th>Language Specifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical Latin</td>
<td>roman</td>
</tr>
<tr>
<td>Croatian</td>
<td>croatian</td>
</tr>
<tr>
<td>Czech</td>
<td>czech</td>
</tr>
<tr>
<td>Danish</td>
<td>danish</td>
</tr>
<tr>
<td>Esperanto</td>
<td>esperanto</td>
</tr>
<tr>
<td>Estonian</td>
<td>estonian</td>
</tr>
<tr>
<td>German phone book order</td>
<td>german2</td>
</tr>
<tr>
<td>Hungarian</td>
<td>hungarian</td>
</tr>
<tr>
<td>Icelandic</td>
<td>icelandic</td>
</tr>
<tr>
<td>Latvian</td>
<td>latvian</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>lithuanian</td>
</tr>
<tr>
<td>Persian</td>
<td>persian</td>
</tr>
<tr>
<td>Polish</td>
<td>polish</td>
</tr>
<tr>
<td>Romanian</td>
<td>romanian</td>
</tr>
<tr>
<td>Sinhala</td>
<td>sinhala</td>
</tr>
<tr>
<td>Slovak</td>
<td>slovak</td>
</tr>
<tr>
<td>Slovenian</td>
<td>slovenian</td>
</tr>
<tr>
<td>Modern Spanish</td>
<td>spanish</td>
</tr>
<tr>
<td>Traditional Spanish</td>
<td>spanish2</td>
</tr>
<tr>
<td>Swedish</td>
<td>swedish</td>
</tr>
<tr>
<td>Turkish</td>
<td>turkish</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>vietnamese</td>
</tr>
</tbody>
</table>

Croatian collations are tailored for these Croatian letters: Č, Ć, Đ, Dž, Lj, Nj, Š, Ž.

Danish collations may also be used for Norwegian.

For Classical Latin collations, I and J compare as equal, and U and V compare as equal.

Spanish collations are available for modern and traditional Spanish. For both, ñ (n-tilde) is a separate letter between n and o. In addition, for traditional Spanish, ch is a separate letter between c and d, and ll is a separate letter between l and m.

Traditional Spanish collations may also be used for Asturian and Galician.

Swedish collations include Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:
Unicode Character Sets

\[ Ú = Y < Ö \]

**general_ci Versus unicode_ci Collations**

For any Unicode character set, operations performed using the `xxx_general_ci` collation are faster than those for the `xxx_unicode_ci` collation. For example, comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`. The reason is that `utf8_unicode_ci` supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, ß is equal to ss in German and some other languages. `utf8_unicode_ci` also supports contractions and ignorable characters. `utf8_general_ci` is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci` (for the effect of this in comparisons or searches, see Section 1.8.6, “Examples of the Effect of Collation”):

\[
\begin{align*}
À &= A \\
Ö &= O \\
Ü &= U \\
ß &= s
\end{align*}
\]

A difference between the collations is that this is true for `utf8_general_ci`:

\[
ß = s
\]

Whereas this is true for `utf8_unicode_ci`, which supports the German DIN-1 ordering (also known as dictionary order):

\[
ß = ss
\]

MySQL implements `utf8` language-specific collations if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German dictionary order and French, so there is no need to create special `utf8` collations.

`utf8_general_ci` also is satisfactory for both German and French, except that ß is equal to s, and not to ss. If this is acceptable for your application, you should use `utf8_general_ci` because it is faster. If this is not acceptable (for example, if you require German dictionary order), use `utf8_unicode_ci` because it is more accurate.

If you require German DIN-2 (phone book) ordering, use the `utf8_german2_ci` collation, which compares the following sets of characters equal:

\[
\begin{align*}
À &= Å = AE \\
Ö &= Æ = OE \\
Ü &= UE \\
ß &= ss
\end{align*}
\]

`utf8_german2_ci` is similar to `latin1_german2_ci`, but the latter does not compare Å equal to AE or Æ equal to OE. There is no `utf8_german_ci` corresponding to `latin1_german_ci` for German dictionary order because `utf8_general_ci` suffices.

**Character Collating Weights**

A character’s collating weight is determined as follows:

- For all Unicode collations except the `bin` (binary) collations, MySQL performs a table lookup to find a character’s collating weight.
Unicode Character Sets

• For _bin collations, the weight is based on the code point, possibly with leading zero bytes added.

Collating weights can be displayed using the WEIGHT_STRING() function. (See String Functions and Operators.) If a collation uses a weight lookup table, but a character is not in the table (for example, because it is a “new” character), collating weight determination becomes more complex:

• For BMP characters in general collations (xxx_general_ci), the weight is the code point.

• For BMP characters in UCA collations (for example, xxx_unicode_ci and language-specific collations), the following algorithm applies:

```sql
if (code >= 0x3400 && code <= 0x4DB5)
    base= 0xFB80; /* CJK Ideograph Extension */
else if (code >= 0x4E00 && code <= 0x9FA5)
    base= 0xFB40; /* CJK Ideograph */
else
    base= 0xFBC0; /* All other characters */
aaa= base + (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

The result is a sequence of two collating elements, `aaa` followed by `bbbb`. For example:

```sql
mysql> SELECT HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci));
+----------------------------------------------------------+
| HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci)) |
+----------------------------------------------------------+
| FBC084CF                                                 |
+----------------------------------------------------------+
```

Thus, `U+04cf CYRILLIC SMALL LETTER PALOCHKA` is, with all UCA 4.0.0 collations, greater than `U +04c0 CYRILLIC LETTER PALOCHKA`. With UCA 5.2.0 collations, all palochkas sort together.

• For supplementary characters in general collations, the weight is the weight for `0xfffd REPLACEMENT CHARACTER`. For supplementary characters in UCA 4.0.0 collations, their collating weight is `0xfffd`. That is, to MySQL, all supplementary characters are equal to each other, and greater than almost all BMP characters.

An example with Deseret characters and COUNT(DISTINCT):

```sql
CREATE TABLE t (s1 VARCHAR(5) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
INSERT INTO t VALUES (0xfffd);   /* REPLACEMENT CHARACTER */
INSERT INTO t VALUES (0x010412); /* DESERET CAPITAL LETTER BEE */
INSERT INTO t VALUES (0x010413); /* DESERET CAPITAL LETTER TEE */
SELECT COUNT(DISTINCT s1) FROM t;
```

The result is 2 because in the MySQL xxx_unicode_ci collations, the replacement character has a weight of `0x0dc6`, whereas Deseret Bee and Deseret Tee both have a weight of `0xfffd`. (Were the utf32_general_ci collation used instead, the result is 1 because all three characters have a weight of `0xfffd` in that collation.)

An example with cuneiform characters and WEIGHT_STRING():

```sql
/*
The four characters in the INSERT string are
0000004A  # LATIN CAPITAL LETTER A
0001218F  # CUNEIFORM SIGN KAB
000121A7  # CUNEIFORM SIGN KISH
00000042  # LATIN CAPITAL LETTER B */
CREATE TABLE t (s1 CHAR(4) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
INSERT INTO t VALUES (0x0000004a0001218f000121a700000042);
SELECT HEX(WIGHT_STRING(s1)) FROM t;
```
The result is:

<table>
<thead>
<tr>
<th>Code</th>
<th>Character</th>
<th>utf8</th>
<th>utf16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0F9D</td>
<td>HALFWIDTH KATAKANA LETTER N</td>
<td>EF BE 9D</td>
<td>FF 9D</td>
</tr>
<tr>
<td>10384</td>
<td>UGARITIC LETTER DELTA</td>
<td>FO 90 8E 84</td>
<td>DF 80 6D 84</td>
</tr>
</tbody>
</table>

The rule that all supplementary characters are equal to each other is nonoptimal but is not expected to cause trouble. These characters are very rare, so it is very rare that a multi-character string consists entirely of supplementary characters. In Japan, since the supplementary characters are obscure Kanji ideographs, the typical user does not care what order they are in, anyway. If you really want rows sorted by the MySQL rule and secondarily by code point value, it is easy:

ORDER BY s1 COLLATE utf32_unicode_ci, s1 COLLATE utf32_bin

- For supplementary characters based on UCA versions higher than 4.0.0 (for example, xxx_unicode_520_ci), supplementary characters do not necessarily all have the same collating weight. Some have explicit weights from the UCA allkeys.txt file. Others have weights calculated from this algorithm:

  aaaa = base + (code >> 15);
  bbbb = (code & 0x7FFF) | 0x8000;

There is a difference between “ordering by the character's code value” and “ordering by the character's binary representation,” a difference that appears only with utf16_bin, because of surrogates.

Suppose that utf16_bin (the binary collation for utf16) was a binary comparison “byte by byte” rather than “character by character.” If that were so, the order of characters in utf16_bin would differ from the order in utf8_bin. For example, the following chart shows two rare characters. The first character is in the range E000-FFFF, so it is greater than a surrogate but less than a supplementary. The second character is a supplementary.

<table>
<thead>
<tr>
<th>Code point</th>
<th>Character</th>
<th>utf8</th>
<th>utf16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0FF9D</td>
<td>HALFWIDTH KATAKANA LETTER N</td>
<td>EF BE 9D</td>
<td>FF 9D</td>
</tr>
<tr>
<td>10384</td>
<td>UGARITIC LETTER DELTA</td>
<td>FO 90 8E 84</td>
<td>DF 80 6D 84</td>
</tr>
</tbody>
</table>

The two characters in the chart are in order by code point value because 0xff9d < 0x10384. And they are in order by utf8 value because 0xef < 0xf0. But they are not in order by utf16 value, if we use byte-by-byte comparison, because 0xff > 0xd8.

So MySQL’s utf16_bin collation is not “byte by byte.” It is “by code point.” When MySQL sees a supplementary-character encoding in utf16, it converts to the character's code-point value, and then compares. Therefore, utf8_bin and utf16_bin are the same ordering. This is consistent with the SQL:2008 standard requirement for a UCS_BASIC collation: “UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted. It is applicable to the UCS character repertoire. Since every character repertoire is a subset of the UCS repertoire, the UCS_BASIC collation is potentially applicable to every character set. NOTE 11: The Unicode scalar value of a character is its code point treated as an unsigned integer.”

If the character set is ucs2, comparison is byte-by-byte, but ucs2 strings should not contain surrogates, anyway.

**Miscellaneous Information**

The xxx_general_mysql500_ci collations preserve the pre-5.1.24 ordering of the original xxx_general_ci collations and permit upgrades for tables created before MySQL 5.1.24 (Bug #27877).
1.10.2 West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- **ascii** (US ASCII) collations:
  - ascii_bin
  - ascii_general_ci (default)
- **cp850** (DOS West European) collations:
  - cp850_bin
  - cp850_general_ci (default)
- **dec8** (DEC Western European) collations:
  - dec8_bin
  - dec8_swedish_ci (default)
- **hp8** (HP Western European) collations:
  - hp8_bin
  - hp8_english_ci (default)
- **latin1** (cp1252 West European) collations:
  - latin1_bin
  - latin1_danish_ci
  - latin1_general_ci
  - latin1_general_cs
  - latin1_german1_ci
  - latin1_german2_ci
  - latin1_spanish_ci
  - latin1_swedish_ci (default)

**latin1** is the default character set. MySQL’s **latin1** is the same as the Windows **cp1252** character set. This means it is the same as the official ISO 8859-1 or IANA (Internet Assigned Numbers Authority) **latin1**, except that IANA **latin1** treats the code points between 0x80 and 0x9f as “undefined,” whereas **cp1252**, and therefore MySQL’s **latin1**, assign characters for those positions. For example, 0x80 is the Euro sign. For the “undefined” entries in **cp1252**, MySQL translates 0x81 to Unicode 0x0081, 0x8d to 0x008d, 0x8f to 0x008f, 0x90 to 0x0090, and 0x9d to 0x009d.

The **latin1_swedish_ci** collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.
The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the “dictionary collation” and DIN-2 is called the “phone book collation.” For an example of the effect this has in comparisons or when doing searches, see Section 1.8.6, “Examples of the Effect of Collation”.

- **latin1_german1_ci** (dictionary) rules:
  - Ä = A
  - Ö = O
  - Ü = U
  - ß = s

- **latin1_german2_ci** (phone-book) rules:
  - Ä = AE
  - Ö = OE
  - Ü = UE
  - ß = ss

In the `latin1_spanish_ci` collation, ñ (n-tilde) is a separate letter between n and o.

- **macroman** (Mac West European) collations:
  - `macroman_bin`
  - `macroman_general_ci` (default)

- **swe7** (7bit Swedish) collations:
  - `swe7_bin`
  - `swe7_swedish_ci` (default)

### 1.10.3 Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- **cp1250** (Windows Central European) collations:
  - `cp1250_bin`
  - `cp1250_croatian_ci`
  - `cp1250_czech_cs`
  - `cp1250_general_ci` (default)
  - `cp1250_polish_ci`

- **cp852** (DOS Central European) collations:
  - `cp852_bin`
  - `cp852_general_ci` (default)

- **keybcs2** (DOS Kamenicky Czech-Slovak) collations:
• keybcs2_bin
• keybcs2_general_ci (default)
• latin2 (ISO 8859-2 Central European) collations:
  • latin2_bin
  • latin2_croatian_ci
  • latin2_czech_cs
  • latin2_general_ci (default)
  • latin2_hungarian_ci
• macce (Mac Central European) collations:
  • macce_bin
  • macce_general_ci (default)

1.10.4 South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

• armSCII8 (ARMSCII-8 Armenian) collations:
  • armSCII8_bin
  • armSCII8_general_ci (default)
• cp1256 (Windows Arabic) collations:
  • cp1256_bin
  • cp1256_general_ci (default)
• geostd8 (GEOSTD8 Georgian) collations:
  • geostd8_bin
  • geostd8_general_ci (default)
• greek (ISO 8859-7 Greek) collations:
  • greek_bin
  • greek_general_ci (default)
• hebrew (ISO 8859-8 Hebrew) collations:
  • hebrew_bin
  • hebrew_general_ci (default)
• latin5 (ISO 8859-9 Turkish) collations:
1.10.5 Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- **cp1257** (Windows Baltic) collations:
  - cp1257_bin
  - cp1257_general_ci (default)
  - cp1257_lithuanian_ci
- **latin7** (ISO 8859-13 Baltic) collations:
  - latin7_bin
  - latin7_estonian_cs
  - latin7_general_ci (default)
  - latin7_general_cs

1.10.6 Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- **cp1251** (Windows Cyrillic) collations:
  - cp1251_bin
  - cp1251_bulgarian_ci
  - cp1251_general_ci (default)
  - cp1251_general_cs
  - cp1251_ukrainian_ci
- **cp866** (DOS Russian) collations:
  - cp866_bin
  - cp866_general_ci (default)
- **koi8r** (KOI8-R Relcom Russian) collations:
  - koi8r_bin
  - koi8r_general_ci (default)
- **koi8u** (KOI8-U Ukrainian) collations:
  - koi8u_bin
1.10.7 Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See Section 1.10.7.1, “The cp932 Character Set”, for additional information about the cp932 and sjis character sets.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets.

- **koi8u_general_ci** (default)

### 1.10.7.1 The cp932 Character Set

- **big5** (Big5 Traditional Chinese) collations:
  - big5_bin
  - big5_chinese_ci (default)

- **cp932** (SJIS for Windows Japanese) collations:
  - cp932_bin
  - cp932_japanese_ci (default)

- **eucjpms** (UJIS for Windows Japanese) collations:
  - eucjpms_bin
  - eucjpms_japanese_ci (default)

- **euckr** (EUC-KR Korean) collations:
  - euckr_bin
  - euckr_korean_ci (default)

- **gb2312** (GB2312 Simplified Chinese) collations:
  - gb2312_bin
  - gb2312_chinese_ci (default)

- **gbk** (GBK Simplified Chinese) collations:
  - gbk_bin
  - gbk_chinese_ci (default)

- **sjis** (Shift-JIS Japanese) collations:
  - sjis_bin
  - sjis_japanese_ci (default)

- **tis620** (TIS620 Thai) collations:
  - tis620_bin
Asian Character Sets

- tis620_thai_ci (default)
- ujis (EUC-JP Japanese) collations:
  - ujis_bin
  - ujis_japanese_ci (default)

The big5_chinese_ci collation sorts on number of strokes.

1.10.7.1 The cp932 Character Set

Why is cp932 needed?

In MySQL, the sjis character set corresponds to the Shift_JIS character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See http://www.iana.org/assignments/character-sets.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to Shift_JIS that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of Shift_JIS and its exact name is Microsoft Windows Codepage : 932 or cp932. In addition to the characters supported by Shift_JIS, cp932 supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (ucs2).
- The sjis character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL cp932 character set is designed to solve these problems.

Because MySQL supports character set conversion, it is important to separate IANA Shift_JIS and cp932 into two different character sets because they provide different conversion rules.

How does cp932 differ from sjis?

The cp932 character set differs from sjis in the following ways:

- cp932 supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some cp932 characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to cp932, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See http://support.microsoft.com/kb/170559/EN-US/.)

The conversion rule works like this:
• If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
• If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
• If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

The table shown at https://msdn.microsoft.com/en-us/goglobal/cc305152.aspx provides information about the Unicode values of \textit{cp932} characters. For \textit{cp932} table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (\textit{ucs2}) encoding. For table entries with an underlined two-digit value appears, there is a range of \textit{cp932} character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the \textit{cp932} characters that begin with those digits.

The following links are of special interest. They correspond to the encodings for the following sets of characters:

• NEC special characters (lead byte 0x87):

• NEC selected—IBM extended characters (lead byte 0xED and 0xEE):

• IBM selected characters (lead byte 0xFA, 0xFB, 0xFC):

• \textit{cp932} supports conversion of user-defined characters in combination with \textit{eucjpms}, and solves the problems with \textit{sjis/ujis} conversion. For details, please refer to http://www.sljfaq.org/afaq/encodings.html.

For some characters, conversion to and from \textit{ucs2} is different for \textit{sjis} and \textit{cp932}. The following tables illustrate these differences.

\textbf{Conversion to \textit{ucs2}}:

<table>
<thead>
<tr>
<th>\textit{sjis/cp932} Value</th>
<th>\textit{sjis} -&gt; \textit{ucs2} Conversion</th>
<th>\textit{cp932} -&gt; \textit{ucs2} Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>5C</td>
<td>005C</td>
<td>005C</td>
</tr>
<tr>
<td>7E</td>
<td>007E</td>
<td>007E</td>
</tr>
<tr>
<td>815C</td>
<td>2015</td>
<td>2015</td>
</tr>
<tr>
<td>815F</td>
<td>005C</td>
<td>FF3C</td>
</tr>
<tr>
<td>8160</td>
<td>301C</td>
<td>FF5E</td>
</tr>
<tr>
<td>8161</td>
<td>2016</td>
<td>2225</td>
</tr>
<tr>
<td>817C</td>
<td>2212</td>
<td>FF0D</td>
</tr>
<tr>
<td>8191</td>
<td>00A2</td>
<td>FFE0</td>
</tr>
<tr>
<td>8192</td>
<td>00A3</td>
<td>FFE1</td>
</tr>
</tbody>
</table>
The Binary Character Set

<table>
<thead>
<tr>
<th>sjis/cp932 Value</th>
<th>sjis -&gt; ucs2 Conversion</th>
<th>cp932 -&gt; ucs2 Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>81CA</td>
<td>00AC</td>
<td>FFE2</td>
</tr>
</tbody>
</table>

Conversion from ucs2:

<table>
<thead>
<tr>
<th>ucs2 value</th>
<th>ucs2 -&gt; sjis Conversion</th>
<th>ucs2 -&gt; cp932 Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>005C</td>
<td>815F</td>
<td>5C</td>
</tr>
<tr>
<td>007E</td>
<td>7E</td>
<td>7E</td>
</tr>
<tr>
<td>00A2</td>
<td>8191</td>
<td>3F</td>
</tr>
<tr>
<td>00A3</td>
<td>8192</td>
<td>3F</td>
</tr>
<tr>
<td>00AC</td>
<td>81CA</td>
<td>3F</td>
</tr>
<tr>
<td>2015</td>
<td>815C</td>
<td>815C</td>
</tr>
<tr>
<td>2016</td>
<td>8161</td>
<td>3F</td>
</tr>
<tr>
<td>2212</td>
<td>817C</td>
<td>3F</td>
</tr>
<tr>
<td>2225</td>
<td>3F</td>
<td>8161</td>
</tr>
<tr>
<td>301C</td>
<td>8160</td>
<td>3F</td>
</tr>
<tr>
<td>FF0D</td>
<td>3F</td>
<td>817C</td>
</tr>
<tr>
<td>FF3C</td>
<td>3F</td>
<td>815F</td>
</tr>
<tr>
<td>FF5E</td>
<td>3F</td>
<td>8160</td>
</tr>
<tr>
<td>FFE0</td>
<td>3F</td>
<td>8191</td>
</tr>
<tr>
<td>FFE1</td>
<td>3F</td>
<td>8192</td>
</tr>
<tr>
<td>FFE2</td>
<td>3F</td>
<td>81CA</td>
</tr>
</tbody>
</table>

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` (or `--skip-character-set-client-handshake`) has an important effect. See Server Command Options.

1.10.8 The Binary Character Set

The binary character set is the character set for binary strings, which are sequences of bytes. The binary character set has one collation, also named binary. Comparison and sorting are based on numeric byte values, rather than on numeric character code values (which for multibyte characters differ from numeric byte values). For information about the differences between the binary collation of the binary character set and the _bin collations of nonbinary character sets, see Section 1.8.5, “The binary Collation Compared to _bin Collations”.

For the binary character set, the concepts of lettercase and accent equivalence do not apply:

- For single-byte characters stored as binary strings, character and byte boundaries are the same, so lettercase and accent differences are significant in comparisons. That is, the binary collation is case-sensitive and accent-sensitive.

```sql
mysql> SET NAMES 'binary';
mysql> SELECT CHARSET('abc'), COLLATION('abc');
+----------------+------------------+
| CHARSET('abc') | COLLATION('abc') |
+----------------+------------------+
| binary         | binary           |
+----------------+------------------+
mysql> SELECT 'abc' = 'ABC', 'a' = 'ä';
```
Restrictions on Character Sets

- For multibyte characters stored as binary strings, character and byte boundaries differ. Character boundaries are lost, so comparisons that depend on them are not meaningful.

To perform lettercase conversion of a binary string, first convert it to a nonbinary string using a character set appropriate for the data stored in the string:

```sql
mysql> SET @str = BINARY 'New York';
+------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
| New York     | new york                           |
+------------------------+
```

To convert a string expression to a binary string, these constructs are equivalent:

```sql
BINARY expr
CAST(expr AS BINARY)
CONVERT(expr USING BINARY)
```

If a value is a character string literal, the `_binary` introducer may be used to designate it as a binary string. For example:

```
_binary 'a'
```

The `_binary` introducer is permitted for hexadecimal literals and bit-value literals as well, but unnecessary; such literals are binary strings by default.

For more information about introducers, see Section 1.3.8, “Character Set Introducers”.

**Note**

Within the `mysql` client, binary strings display using hexadecimal notation, depending on the value of the `--binary-as-hex` option, see `mysql — The MySQL Command-Line Client`.

### 1.11 Restrictions on Character Sets

- Identifiers are stored in `mysql` database tables (`user`, `db`, and so forth) using `utf8`, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.

- The `ucs2`, `utf16`, `utf16le`, and `utf32` character sets have the following restrictions:
  - None of them can be used as the client character set. See Impermissible Client Character Sets.
  - It is currently not possible to use `LOAD DATA` to load data files that use these character sets.
  - `FULLTEXT` indexes cannot be created on a column that uses any of these character sets. However, you can perform `IN BOOLEAN MODE` searches on the column without an index.
  - The use of `ENCRYPT()` with these character sets is not recommended because the underlying system call expects a string terminated by a zero byte.
• The **REGEXP** and **RLIKE** operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

### 1.12 Setting the Error Message Language

By default, `mysqld` produces error messages in English, but they can be displayed instead in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish. This applies to messages the server writes to the error log and sends to clients.

To select the language in which the server writes error messages, follow the instructions in this section. For information about changing the character set for error messages (rather than the language), see Section 1.6, “Error Message Character Set”. For general information about configuring error logging, see The Error Log.

The server searches for the error message file using these rules:

• It looks for the file in a directory constructed from two system variable values, `lc_messages_dir` and `lc_messages`, with the latter converted to a language name. Suppose that you start the server using this command:

```
mysqld --lc_messages_dir=/usr/share/mysql --lc_messages=fr_FR
```

In this case, `mysqld` maps the locale `fr_FR` to the language `french` and looks for the error file in the `usr/share/mysql/french` directory.

By default, the language files are located in the `share/mysql/LANGUAGE` directory under the MySQL base directory.

• If the message file cannot be found in the directory constructed as just described, the server ignores the `lc_messages` value and uses only the `lc_messages_dir` value as the location in which to look.

The `lc_messages_dir` system variable can be set only at server startup and has only a global read-only value at runtime. `lc_messages` can be set at server startup and has global and session values that can be modified at runtime. Thus, the error message language can be changed while the server is running, and each client can have its own error message language by setting its session `lc_messages` value to the desired locale name. For example, if the server is using the `fr_FR` locale for error messages, a client can execute this statement to receive error messages in English:

```
SET lc_messages = 'en_US';
```

### 1.13 Adding a Character Set

This section discusses the procedure for adding a character set to MySQL. The proper procedure depends on whether the character set is simple or complex:

• If the character set does not need special string collating routines for sorting and does not need multibyte character support, it is simple.

• If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.
Adding a Character Set

To use the following instructions, you must have a MySQL source distribution. In the instructions, \textit{MYSET} represents the name of the character set that you want to add.

1. Add a \texttt{<charset>} element for \textit{MYSET} to the \texttt{sql/share/charsets/Index.xml} file. Use the existing contents in the file as a guide to adding new contents. A partial listing for the \texttt{latin1} \texttt{<charset>} element follows:

   ```xml
   <charset name="latin1">
     <family>Western</family>
     <description>cp1252 West European</description>
     ...
     <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
       <flag>primary</flag>
       <flag>compiled</flag>
     </collation>
     ...
     <collation name="latin1_danish_ci" id="15" order="Danish"/>
     ...
     <collation name="latin1_bin" id="47" order="Binary">
       <flag>binary</flag>
       <flag>compiled</flag>
     </collation>
     ...
   </charset>
   ```

   The \texttt{<charset>} element must list all the collations for the character set. These must include at least a binary collation and a default (primary) collation. The default collation is often named using a suffix of \texttt{general_ci} (general, case-insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a \texttt{primary} flag. The binary collation should have a \texttt{binary} flag.

   You must assign a unique ID number to each collation. The range of IDs from 1024 to 2047 is reserved for user-defined collations. To find the maximum of the currently used collation IDs, use this query:

   ```sql
   SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
   ```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multibyte functions, or both.

   For a simple character set, create a configuration file, \texttt{MYSET.xml}, that describes the character set properties. Create this file in the \texttt{sql/share/charsets} directory. You can use a copy of \texttt{latin1.xml} as the basis for this file. The syntax for the file is very simple:

   - Comments are written as ordinary XML comments (\texttt{<!-- text -->}).
   - Words within \texttt{<map>} array elements are separated by arbitrary amounts of whitespace.
   - Each word within \texttt{<map>} array elements must be a number in hexadecimal format.
   - The \texttt{<map>} array element for the \texttt{<ctype>} element has 257 words. The other \texttt{<map>} array elements after that have 256 words. See Section 1.13.1, "Character Definition Arrays".
   - For each collation listed in the \texttt{<charset>} element for the character set in \texttt{Index.xml}, \texttt{MYSET.xml} must contain a \texttt{<collation>} element that defines the character ordering.

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   - Create the file \texttt{ctype-MYSET.c} in the \texttt{strings} directory. Look at one of the existing \texttt{ctype-*.c} files (such as \texttt{ctype-big5.c}) to see what needs to be defined. The arrays in your file must have
names like `ctype_MYSET`, `to_lower_MYSET`, and so on. These correspond to the arrays for a simple character set. See Section 1.13.1, “Character Definition Arrays”.

- For each `<collation>` element listed in the `<charset>` element for the character set in `Index.xml`, the `ctype-MYSET.c` file must provide an implementation of the collation.

- If the character set requires string collating functions, see Section 1.13.2, “String Collating Support for Complex Character Sets”.

- If the character set requires multibyte character support, see Section 1.13.3, “Multi-Byte Character Support for Complex Character Sets”.

3. Modify the configuration information. Use the existing configuration information as a guide to adding information for `MYSYS`. The example here assumes that the character set has default and binary collations, but more lines are needed if `MYSET` has additional collations.

   a. Edit `mysys/charset-def.c`, and “register” the collations for the new character set.

   Add these lines to the “declaration” section:

   ```c
   #ifdef HAVE_CHARSET_MYSET
   extern CHARSET_INFO my_charset_MYSET_general_ci;
   extern CHARSET_INFO my_charset_MYSET_bin;
   #endif
   
   Add these lines to the “registration” section:
   ```
   ```
   #ifdef HAVE_CHARSET_MYSET
   add_compiled_collation(&my_charset_MYSET_general_ci);
   add_compiled_collation(&my_charset_MYSET_bin);
   #endif
   ```

   b. If the character set uses `ctype-MYSET.c`, edit `strings/CMakeLists.txt` and add `ctype-MYSET.c` to the definition of the `STRINGS_SOURCES` variable.

   c. Edit `cmake/character_sets.cmake`:

      i. Add `MYSET` to the value of with `CHARSETS_AVAILABLE` in alphabetic order.

      ii. Add `MYSET` to the value of `CHARSETS_COMPLEX` in alphabetic order. This is needed even for simple character sets, or `CMake` does not recognize `-DDEFAULT_CHARSET=MYSET`

4. Reconfigure, recompile, and test.

### 1.13.1 Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. For a character set named `MYSYS`, the file is named `MYSET.xml`. It uses `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character.

- `<lower>` and `<upper>` list the lowercase and uppercase characters.

- `<unicode>` maps 8-bit character values to Unicode values.

- `<collation>` elements indicate character ordering for comparison and sorting, one element per collation. Binary collations need no `<map>` element because the character codes themselves provide the ordering.
String Collating Support for Complex Character Sets

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See also the existing `ctype-*.c` files for examples. See the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

Most of the arrays are indexed by character value and have 256 elements. The `<ctype>` array is indexed by character value + 1 and has 257 elements. This is a legacy convention for handling `EOF`.

`<ctype>` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m ctype.h`:

```c
#define _MY_U  01      /* Upper case */
#define _MY_L  02      /* Lower case */
#define _MY_NMR 04      /* Numeral (digit) */
#define _MY_SPC 010     /* Spacing character */
#define _MY_PNT 020     /* Punctuation */
#define _MY_CTR 040     /* Control character */
#define _MY_B   0100    /* Blank */
#define _MY_X   0200    /* heXadecimal digit */
```

The `<ctype>` value for a given character should be the union of the applicable bitmask values that describe the character. For example, 'A' is an uppercase character (_MY_U) as well as a hexadecimal digit (_MY_X), so its `ctype` value should be defined like this:

```
cctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m ctype.h` are octal values, but the elements of the `<ctype>` array in `MYSET.xml` should be written as hexadecimal values.

The `<lower>` and `<upper>` arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```c
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each `<collation>` array indicates how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the `<upper>` array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in Section 1.13.2, “String Collating Support for Complex Character Sets”.

### 1.13.2 String Collating Support for Complex Character Sets

For a simple character set named `MYSET`, sorting rules are specified in the `MYSET.xml` configuration file using `<map>` array elements within `<collation>` elements. If the sorting rules for your language are too complex to be handled with simple arrays, you must define string collating functions in the `ctype-MYSET.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets. Take a look at the `MY_COLLATION_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

### 1.13.3 Multi-Byte Character Support for Complex Character Sets

If you want to add support for a new character set named `MYSET` that includes multibyte characters, you must use multibyte character functions in the `ctype-MYSET.c` source file in the `strings` directory.
Adding a Collation to a Character Set

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the *ctype-* files in the strings directory, such as the files for the euc_kr, gb2312, gbk, sjis, and ujis character sets. Take a look at the MY_CHARSET_HANDLER structures to see how they are used. See also the CHARSET_INFO.txt file in the strings directory for additional information.

1.14 Adding a Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

The WEIGHT_STRING() function can be used to see the weights for the characters in a string. The value that it returns to indicate weights is a binary string, so it is convenient to use HEX(WEIGHT_STRING(str)) to display the weights in printable form. The following example shows that weights do not differ for lettercase for the letters in 'AaBb' if it is a nonbinary case-insensitive string, but do differ if it is a binary string:

```sql
mysql> SELECT HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci));
+------------------------------------------------------+
| HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci)) |
+------------------------------------------------------+
| 41414242                                             |
+------------------------------------------------------+
mysql> SELECT HEX(WEIGHT_STRING(BINARY 'AaBb'));
+-----------------------------------+
| HEX(WEIGHT_STRING(BINARY 'AaBb')) |
+-----------------------------------+
| 41614262                          |
+-----------------------------------+
```

MySQL supports several collation implementations, as discussed in Section 1.14.1, “Collation Implementation Types”. Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets.
- UCA-based collations for Unicode character sets.
- Binary (xxx_bin) collations.

The following sections describe how to add user-defined collations of the first two types to existing character sets. All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new user-defined collation:

1. Choose a collation ID.
2. Add configuration information that names the collation and describes the character-ordering rules.
3. Restart the server.
4. Verify that the server recognizes the collation.

The instructions here cover only user-defined collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source
file), use the instructions in Section 1.13, “Adding a Character Set”. However, instead of adding all the
information required for a complete character set, just modify the appropriate files for an existing character
set. That is, based on what is already present for the character set's current collations, add data structures,
functions, and configuration information for the new collation.

Note

If you modify an existing user-defined collation, that may affect the ordering of rows
for indexes on columns that use the collation. In this case, rebuild any such indexes
to avoid problems such as incorrect query results. See Rebuilding or Repairing
Tables or Indexes.

Additional Resources

- Example showing how to add a collation for full-text searches: Adding a User-Defined Collation for Full-
  Text Indexing
- The Unicode Collation Algorithm (UCA) specification: http://www.unicode.org/reports/tr10/
- The Locale Data Markup Language (LDML) specification: http://www.unicode.org/reports/tr35/

1.14.1 Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from
character codes to weights. latin1_swedish_ci is an example. It is a case-insensitive collation, so the
uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX(WEIGHT_STRING('a')), HEX(WEIGHT_STRING('A'));
+-------------------------+-------------------------+
| HEX(WEIGHT_STRING('a')) | HEX(WEIGHT_STRING('A')) |
+-------------------------+-------------------------+
| 41                      | 41                      |
+-------------------------+-------------------------+
1 row in set (0.01 sec)
```

For implementation instructions, see Section 1.14.3, “Adding a Simple Collation to an 8-Bit Character Set”.

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters,
as described in Section 1.13, “Adding a Character Set”.

Collations for non-Unicode multibyte character sets

For this type of collation, 8-bit (single-byte) and multibyte characters are handled differently. For 8-bit
characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte
characters 'a' and 'A' both have a weight of 0x41.) For multibyte characters, there are two types of
relationship between character codes and weights:
Collation Implementation Types

- Weights equal character codes. `sjis_japanese_ci` is an example of this kind of collation. The multibyte character ‘ぢ’ has a character code of 0x82C0, and the weight is also 0x82C0.

```sql
mysql> CREATE TABLE t1
     > (c1 VARCHAR(2) CHARACTER SET sjis COLLATE sjis_japanese_ci);
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x82C0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
| a    | 61      | 41                     |
| A    | 41      | 41                     |
| ぢ    | 82C0    | 82C0                   |
+------+---------+------------------------+
3 rows in set (0.00 sec)
```

- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. `gbk_chinese_ci` is an example of this kind of collation. The multibyte character ‘膰’ has a character code of 0x81B0 but a weight of 0xC286.

```sql
mysql> CREATE TABLE t1
     > (c1 VARCHAR(2) CHARACTER SET gbk COLLATE gbk_chinese_ci);
Query OK, 0 rows affected (0.33 sec)
mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x81B0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
| a    | 61      | 41                     |
| A    | 41      | 41                     |
| 膰    | 81B0    | C286                   |
+------+---------+------------------------+
3 rows in set (0.00 sec)
```

For implementation instructions, see Section 1.13, “Adding a Character Set”.

Collations for Unicode multibyte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case-insensitive and accent-insensitive. `utf8_general_ci` is an example: 'a', 'A', 'À', and 'á' each have different character codes but all have a weight of 0x0041 and compare as equal.

```sql
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE TABLE t1
     > (c1 CHAR(1) CHARACTER SET UTF8 COLLATE utf8_general_ci);
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO t1 VALUES ('a'),('A'),('À'),('á');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0
mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
| a    | 61      | 0041                   |
| A    | 41      | 0041                   |
| À    | C380    | 0041                   |
| á    | C3A1    | 0041                   |
+------+---------+------------------------+
4 rows in set (0.00 sec)
```
Choosing a Collation ID

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits).
- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: 'a' has a weight of 0x0E33.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.05 sec)
mysql> SELECT HEX('a'), HEX(WEIGHT_STRING('a'));
+----------+-------------------------+
| HEX('a') | HEX(WEIGHT_STRING('a')) |
|-----------+--------------------------|
| 61        | 0E33                     |
+----------+-------------------------+
1 row in set (0.02 sec)
```

- A character may have many weights. This is an expansion. Example: The German letter 'ß' (SZ ligature, or SHARP S) has a weight of 0x0FEA0FEA.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.11 sec)
mysql> SELECT HEX('ß'), HEX(WEIGHT_STRING('ß'));
+-----------+--------------------------+
| HEX('ß')  | HEX(WEIGHT_STRING('ß'))  |
|-----------+--------------------------|
| C39F      | 0FEA0FEA                 |
+-----------+--------------------------+
1 row in set (0.00 sec)
```

- Many characters may have one weight. This is a contraction. Example: 'ch' is a single letter in Czech and has a weight of 0x0EE2.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_czech_ci';
Query OK, 0 rows affected (0.09 sec)
mysql> SELECT HEX('ch'), HEX(WEIGHT_STRING('ch'));
+-----------+--------------------------+
| HEX('ch') | HEX(WEIGHT_STRING('ch')) |
|-----------+--------------------------|
| 6368      | 0EE2                     |
+-----------+--------------------------+
1 row in set (0.00 sec)
```

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

For implementation instructions, for a non-UCA collation, see Section 1.13, “Adding a Character Set”. For a UCA collation, see Section 1.14.4, “Adding a UCA Collation to a Unicode Character Set”.

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

1.14.2 Choosing a Collation ID

Each collation must have a unique ID. To add a collation, you must choose an ID value that is not currently used. MySQL supports two-byte collation IDs. The range of IDs from 1024 to 2047 is reserved for user-defined collations.
The collation ID that you choose appears in these contexts:

- The ID column of the INFORMATION_SCHEMA.COLLATIONS table.
- The Id column of SHOW COLLATION output.
- The charsetnr member of the MYSQL_FIELD C API data structure.
- The number member of the MY_CHARSET_INFO data structure returned by the mysql_get_character_set_info() C API function.

To determine the largest currently used ID, issue the following statement:

```sql
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+---------+
<table>
<thead>
<tr>
<th>MAX(ID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>247</td>
</tr>
</tbody>
</table>
+---------+
```

To display a list of all currently used IDs, issue this statement:

```sql
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+-----+
<table>
<thead>
<tr>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>52</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>57</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>98</td>
</tr>
<tr>
<td>99</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>129</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>247</td>
</tr>
</tbody>
</table>
+-----+
```

**Warning**

Before upgrading, you should save the configuration files that you change. If you upgrade in place, the process replaces the modified files.

### 1.14.3 Adding a Simple Collation to an 8-Bit Character Set

This section describes how to add a simple collation for an 8-bit character set by writing the `<collation>` elements associated with a `<charset>` character set description in the MySQL Index.xml file. The procedure described here does not require recompiling MySQL. The example adds a collation named latin1_test_ci to the latin1 character set.

1. Choose a collation ID, as shown in Section 1.14.2, "Choosing a Collation ID". The following steps use an ID of 1024.

2. Modify the Index.xml and latin1.xml configuration files. These files are located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:
Adding a UCA Collation to a Unicode Character Set

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. For example:

```xml
<charset name="latin1">
  ...
  <collation name="latin1_test_ci" id="1024"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table for character codes 0 to 255. Each value within the `<map>` element must be a number in hexadecimal format.

```xml
<collation name="latin1_test_ci">
  <map>
    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
    10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
    20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
    30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
    40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
    50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
    60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F
    70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F
  </map>
</collation>
```

5. Restart the server and use this statement to verify that the collation is present:

```sql
mysql> SHOW COLLATION WHERE Collation = 'latin1_test_ci';
+----------------+---------+------+---------+----------+---------+
| Collation      | Charset | Id   | Default | Compiled | Sortlen |
+----------------+---------+------+---------+----------+---------+
| latin1_test_ci | latin1  | 1024 |         |          |       1 |
+----------------+---------+------+---------+----------+---------+
```

1.14.4 Adding a UCA Collation to a Unicode Character Set

This section describes how to add a UCA collation for a Unicode character set by writing the `<collation>` element within a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. It uses a subset of the Locale Data Markup Language (LDML) specification, which is available at [http://www.unicode.org/reports/tr35/](http://www.unicode.org/reports/tr35/). With this method, you need not define the entire collation. Instead, you begin with an existing “base” collation and describe the new collation in terms of how it differs from the base collation. The following table lists the base collations of the Unicode character sets for which UCA collations can be defined. It is not possible to create user-defined UCA collations for `utf16le`; there is no `utf16le_unicode_ci` collation that would serve as the basis for such collations.
### Table 1.4 MySQL Character Sets Available for User-Defined UCA Collations

<table>
<thead>
<tr>
<th>Character Set</th>
<th>Base Collation</th>
</tr>
</thead>
<tbody>
<tr>
<td>utf8</td>
<td>utf8_unicode_ci</td>
</tr>
<tr>
<td>ucs2</td>
<td>ucs2_unicode_ci</td>
</tr>
<tr>
<td>utf16</td>
<td>utf16_unicode_ci</td>
</tr>
<tr>
<td>utf32</td>
<td>utf32_unicode_ci</td>
</tr>
</tbody>
</table>

The following sections show how to add a collation that is defined using LDML syntax, and provide a summary of LDML rules supported in MySQL.

#### 1.14.4.1 Defining a UCA Collation Using LDML Syntax

To add a UCA collation for a Unicode character set without recompiling MySQL, use the following procedure. If you are unfamiliar with the LDML rules used to describe the collation's sort characteristics, see Section 1.14.4.2, “LDML Syntax Supported in MySQL”.

The example adds a collation named `utf8_phone_ci` to the `utf8` character set. The collation is designed for a scenario involving a Web application for which users post their names and phone numbers. Phone numbers can be given in very different formats:

```
+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567
```

The problem raised by dealing with these kinds of values is that the varying permissible formats make searching for a specific phone number very difficult. The solution is to define a new collation that reorders punctuation characters, making them ignorable.

1. Choose a collation ID, as shown in Section 1.14.2, “Choosing a Collation ID”. The following steps use an ID of 1029.

2. To modify the `Index.xml` configuration file. This file is located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

   ```
   mysql> SHOW VARIABLES LIKE 'character_sets_dir';
   +--------------------+-----------------------------------------+
   | Variable_name      | Value                                   |
   +--------------------+-----------------------------------------+
   | character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
   +--------------------+-----------------------------------------+
   ```

3. Choose a name for the collation and list it in the `Index.xml` file. In addition, you'll need to provide the collation ordering rules. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. Within the `<collation>` element, provide a `<rules>` element containing the ordering rules:

   ```xml
   <charset name="utf8">
   ...
   <collation name="utf8_phone_ci" id="1029">
   <rules>
   <reset> </reset>
   <i> </i> <!-- space -->
   <i>(</i> <!-- left parenthesis -->
   <i>)</i> <!-- right parenthesis -->
   </rules>
   </collation>
   ```
Adding a UCA Collation to a Unicode Character Set

4. If you want a similar collation for other Unicode character sets, add other `<collation>` elements. For example, to define `ucs2_phone_ci`, add a `<collation>` element to the `<charset name="ucs2">` element. Remember that each collation must have its own unique ID.

5. Restart the server and use this statement to verify that the collation is present:

```sql
mysql> SHOW COLLATION WHERE Collation = 'utf8_phone_ci';
+---------------+---------+------+---------+----------+---------+
| Collation     | Charset | Id   | Default | Compiled | Sortlen |
|---------------+---------+------+---------+----------+---------+
| utf8_phone_ci | utf8    | 1029 |         |          |       8 |
+---------------+---------+------+---------+----------+---------+
```

Now test the collation to make sure that it has the desired properties.

Create a table containing some sample phone numbers using the new collation:

```sql
mysql> CREATE TABLE phonebook (|
    | name VARCHAR(64),
    | phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci|
    |); Query OK, 0 rows affected (0.09 sec)
```

```sql
mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02'); Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04'); Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01'); Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03'); Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005'); Query OK, 1 row affected (0.00 sec)
```

Run some queries to see whether the ignored punctuation characters are in fact ignored for comparison and sorting:

```sql
mysql> SELECT * FROM phonebook ORDER BY phone;
+---------+------------------+
| name    | phone            |
+---------+------------------+
| Sanja   | +380 (912) 8008005|
| Bar     | +7-912-800-80-01 |
| Svoj    | +7 912 800 80 02 |
| Ramil   | (7912) 800 80 03 |
| Hf      | +7 (912) 800 80 04|
+---------+------------------+
5 rows in set (0.00 sec)
```

```sql
mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
+---------+------------------+
| name    | phone            |
+---------+------------------+
| Bar     | +7-912-800-80-01 |
+---------+------------------+
1 row in set (0.00 sec)
```

```sql
mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+---------+------------------+
| name    | phone            |
+---------+------------------+
+---------+------------------+
```

69
1.14.4.2 LDML Syntax Supported in MySQL

This section describes the LDML syntax that MySQL recognizes. This is a subset of the syntax described in the LDML specification available at http://www.unicode.org/reports/tr35/, which should be consulted for further information. MySQL recognizes a large enough subset of the syntax that, in many cases, it is possible to download a collation definition from the Unicode Common Locale Data Repository and paste the relevant part (that is, the part between the `<rules>` and `</rules>` tags) into the MySQL `Index.xml` file. The rules described here are all supported except that character sorting occurs only at the primary level. Rules that specify differences at secondary or higher sort levels are recognized (and thus can be included in collation definitions) but are treated as equality at the primary level.

The MySQL server generates diagnostics when it finds problems while parsing the `Index.xml` file. See Section 1.14.4.3, “Diagnostics During Index.xml Parsing”.

Character Representation

Characters named in LDML rules can be written literally or in \u{nnnn} format, where \( nnnn \) is the hexadecimal Unicode code point value. For example, \( \text{A} \) and \( \text{á} \) can be written literally or as \( \text{\u{0041}} \) and \( \text{\u{00E1}} \). Within hexadecimal values, the digits \( \text{A} \) through \( \text{F} \) are not case-sensitive; \( \text{\u{00E1}} \) and \( \text{\u{00e1}} \) are equivalent. For UCA 4.0.0 collations, hexadecimal notation can be used only for characters in the Basic Multilingual Plane, not for characters outside the BMP range of \( \text{0000 to FFFF} \). For UCA 5.2.0 collations, hexadecimal notation can be used for any character.

The `Index.xml` file itself should be written using UTF-8 encoding.

Syntax Rules

LDML has reset rules and shift rules to specify character ordering. Orderings are given as a set of rules that begin with a reset rule that establishes an anchor point, followed by shift rules that indicate how characters sort relative to the anchor point.

- A `<reset>` rule does not specify any ordering in and of itself. Instead, it “resets” the ordering for subsequent shift rules to cause them to be taken in relation to a given character. Either of the following rules resets subsequent shift rules to be taken in relation to the letter \( \text{A} \):

  
  ```xml
  <reset>A</reset>
  <reset>\u{0041}</reset>
  ```

- The `<p>`, `<s>`, and `<t>` shift rules define primary, secondary, and tertiary differences of a character from another character:

  - Use primary differences to distinguish separate letters.
  - Use secondary differences to distinguish accent variations.
  - Use tertiary differences to distinguish lettercase variations.

    Either of these rules specifies a primary shift rule for the \( \text{G} \) character:
• The <i> shift rule indicates that one character sorts identically to another. The following rules cause 'b' to sort the same as 'a':

```reset</reset>a</i><b/>
```

• Abbreviated shift syntax specifies multiple shift rules using a single pair of tags. The following table shows the correspondence between abbreviated syntax rules and the equivalent nonabbreviated rules.

Table 1.5 Abbreviated Shift Syntax

<table>
<thead>
<tr>
<th>Abbreviated Syntax</th>
<th>Nonabbreviated Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;pc&gt;xyz&lt;/pc&gt;</td>
<td>&lt;p&gt;x&lt;/p&gt;&lt;p&gt;y&lt;/p&gt;&lt;p&gt;z&lt;/p&gt;</td>
</tr>
<tr>
<td>&lt;sc&gt;xyz&lt;/sc&gt;</td>
<td>&lt;s&gt;x&lt;/s&gt;&lt;s&gt;y&lt;/s&gt;&lt;s&gt;z&lt;/s&gt;</td>
</tr>
<tr>
<td>&lt;tc&gt;xyz&lt;/tc&gt;</td>
<td>&lt;t&gt;x&lt;/t&gt;&lt;t&gt;y&lt;/t&gt;&lt;t&gt;z&lt;/t&gt;</td>
</tr>
<tr>
<td>&lt;ic&gt;xyz&lt;/ic&gt;</td>
<td>&lt;i&gt;x&lt;/i&gt;&lt;i&gt;y&lt;/i&gt;&lt;i&gt;z&lt;/i&gt;</td>
</tr>
</tbody>
</table>

• An expansion is a reset rule that establishes an anchor point for a multiple-character sequence. MySQL supports expansions 2 to 6 characters long. The following rules put 'z' greater at the primary level than the sequence of three characters 'abc':

```reset</reset>abc</reset><p>z</p>
```

• A contraction is a shift rule that sorts a multiple-character sequence. MySQL supports contractions 2 to 6 characters long. The following rules put the sequence of three characters 'xyz' greater at the primary level than 'a':

```reset</reset>a</reset><p>xyz</p>
```

• Long expansions and long contractions can be used together. These rules put the sequence of three characters 'xyz' greater at the primary level than the sequence of three characters 'abc':

```reset</reset>abc</reset><p>xyz</p>
```

• Normal expansion syntax uses <x> plus <extend> elements to specify an expansion. The following rules put the character 'k' greater at the secondary level than the sequence 'ch'. That is, 'k' behaves as if it expands to a character after 'c' followed by 'h':

```reset</reset>c</reset><x>s<k</x>s<extend>h</extend></x>
```

This syntax permits long sequences. These rules sort the sequence 'ccs' greater at the tertiary level than the sequence 'cscs':

```reset</reset>cs</reset><x>t<ccs</x>t<extend>cs</extend></x>
```

The LDML specification describes normal expansion syntax as “tricky.” See that specification for details.

• Previous context syntax uses <x> plus <context> elements to specify that the context before a character affects how it sorts. The following rules put '−' greater at the secondary level than 'a', but only when '−' occurs after 'b':

```reset</reset>−</reset><x>a</x><context>b</context><extend>−</extend></x>
Adding a UCA Collation to a Unicode Character Set

• Previous context syntax can include the <extend> element. These rules put 'def' greater at the primary level than 'aghi', but only when 'def' comes after 'abc':

```
<reset>a</reset>
<x><context>b</context><x>s</x><s>-</s><x>s</x></x>
```

• Reset rules permit a before attribute. Normally, shift rules after a reset rule indicate characters that sort after the reset character. Shift rules after a reset rule that has the before attribute indicate characters that sort before the reset character. The following rules put the character 'b' immediately before 'a' at the primary level:

```
<reset before="primary">a</reset>
<p>b</p>
```

Permissible before attribute values specify the sort level by name or the equivalent numeric value:

```
<reset before="primary">
<reset before="1">
<reset before="secondary">
<reset before="2">
<reset before="tertiary">
<reset before="3">
```

• A reset rule can name a logical reset position rather than a literal character:

```
<first_tertiary_ignorable/>
<last_tertiary_ignorable/>
<first_secondary_ignorable/>
<last_secondary_ignorable/>
<first_primary_ignorable/>
<last_primary_ignorable/>
<first_variable/>
<last_variable/>
<first_non_ignorable/>
<last_non_ignorable/>
<first_trailing/>
<last_trailing/>
```

These rules put 'z' greater at the primary level than nonignorable characters that have a Default Unicode Collation Element Table (DUCET) entry and that are not CJK:

```
<reset><last_non_ignorable/></reset>
<p>z</p>
```

Logical positions have the code points shown in the following table.

### Table 1.6 Logical Reset Position Code Points

<table>
<thead>
<tr>
<th>Logical Position</th>
<th>Unicode 4.0.0 Code Point</th>
<th>Unicode 5.2.0 Code Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;first_non_ignorable/&gt;</td>
<td>U+02D0</td>
<td>U+02D0</td>
</tr>
<tr>
<td>&lt;last_non_ignorable/&gt;</td>
<td>U+A48C</td>
<td>U+1342E</td>
</tr>
<tr>
<td>&lt;first_primary_ignorable/&gt;</td>
<td>U+0332</td>
<td>U+0332</td>
</tr>
<tr>
<td>&lt;last_primary_ignorable/&gt;</td>
<td>U+0332</td>
<td>U+0332</td>
</tr>
<tr>
<td>&lt;first_secondary_ignorable/&gt;</td>
<td>U+20EA</td>
<td>U+101FD</td>
</tr>
<tr>
<td>&lt;first_non_ignorable/&gt;</td>
<td>U+0000</td>
<td>U+0000</td>
</tr>
</tbody>
</table>
### Adding a UCA Collation to a Unicode Character Set

#### Logical Position

<table>
<thead>
<tr>
<th>Element</th>
<th>Unicode 4.0.0 Code Point</th>
<th>Unicode 5.2.0 Code Point</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;last_secondary_ignorable/&gt;</code></td>
<td>U+FE73</td>
<td>U+FE73</td>
</tr>
<tr>
<td><code>&lt;first_tertiary_ignorable/&gt;</code></td>
<td>U+0000</td>
<td>U+0000</td>
</tr>
<tr>
<td><code>&lt;last_tertiary_ignorable/&gt;</code></td>
<td>U+FE73</td>
<td>U+FE73</td>
</tr>
<tr>
<td><code>&lt;first_trailing/&gt;</code></td>
<td>U+0000</td>
<td>U+0000</td>
</tr>
<tr>
<td><code>&lt;last_trailing/&gt;</code></td>
<td>U+0000</td>
<td>U+0000</td>
</tr>
<tr>
<td><code>&lt;first_variable/&gt;</code></td>
<td>U+0009</td>
<td>U+0009</td>
</tr>
<tr>
<td><code>&lt;last_variable/&gt;</code></td>
<td>U+2183</td>
<td>U+1D371</td>
</tr>
</tbody>
</table>

- The `<collation>` element permits a `shift-after-method` attribute that affects character weight calculation for shift rules. The attribute has these permitted values:
  - **simple**: Calculate character weights as for reset rules that do not have a `before` attribute. This is the default if the attribute is not given.
  - **expand**: Use expansions for shifts after reset rules.

Suppose that '0' and '1' have weights of 0E29 and 0E2A and we want to put all basic Latin letters between '0' and '1':

```xml
<reset>0</reset>
<pc>abcdefghijklmnopqrstuvwxyz</pc>
```

For simple shift mode, weights are calculated as follows:

- 'a' has weight 0E29+1
- 'b' has weight 0E29+2
- 'c' has weight 0E29+3
- ...

However, there are not enough vacant positions to put 26 characters between '0' and '1'. The result is that digits and letters are intermixed.

To solve this, use `shift-after-method="expand"`. Then weights are calculated like this:

- 'a' has weight [0E29][233D+1]
- 'b' has weight [0E29][233D+2]
- 'c' has weight [0E29][233D+3]
- ...

233D is the UCA 4.0.0 weight for character 0xA48C, which is the last nonignorable character (a sort of the greatest character in the collation, excluding CJK). UCA 5.2.0 is similar but uses 3ACA, for character 0x1342E.

### MySQL-Specific LDML Extensions

An extension to LDML rules permits the `<collation>` element to include an optional `version` attribute in `<collation>` tags to indicate the UCA version on which the collation is based. If the `version` attribute is omitted, its default value is 4.0.0. For example, this specification indicates a collation that is based on UCA 5.2.0:

```xml
<collation id="nnn" name="utf8_xxx_ci" version="5.2.0">...
```
1.14.4.3 Diagnostics During Index.xml Parsing

The MySQL server generates diagnostics when it finds problems while parsing the Index.xml file:

- Unknown tags are written to the error log. For example, the following message results if a collation definition contains a `<aaa>` tag:

  [Warning] Buffered warning: Unknown LDML tag: 'charsets/charset/collation/rules/aaa'

- If collation initialization is not possible, the server reports an “Unknown collation” error, and also generates warnings explaining the problems, such as in the previous example. In other cases, when a collation description is generally correct but contains some unknown tags, the collation is initialized and is available for use. The unknown parts are ignored, but a warning is generated in the error log.

- Problems with collations generate warnings that clients can display with SHOW WARNINGS. Suppose that a reset rule contains an expansion longer than the maximum supported length of 6 characters:

  `<reset>abcdefghi</reset>`
  `<i>x</i>`

  An attempt to use the collation produces warnings:

      mysql> SELECT _utf8'test' COLLATE utf8_test_ci;
      ERROR 1273 (HY000): Unknown collation: 'utf8_test_ci'
      mysql> SHOW WARNINGS;

      +---------+------+----------------------------------------+
      | Level   | Code | Message                                |
      +---------+------+----------------------------------------+
      | Error   | 1273 | Unknown collation: 'utf8_test_ci'      |
      | Warning | 1273 | Expansion is too long at 'abcdefghi=x' |
      +---------+------+----------------------------------------+

1.15 Character Set Configuration

The MySQL server has a compiled-in default character set and collation. To change these defaults, use the `--character-set-server` and `--collation-server` options when you start the server. See Server Command Options. The collation must be a legal collation for the default character set. To determine which collations are available for each character set, use the SHOW COLLATION statement or query the INFORMATION_SCHEMA COLLATIONS table.

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- If your program uses an incorrect path to determine where the character sets are stored (which is typically the share/mysql/charsets or share/charsets directory under the MySQL installation directory), this can be fixed by using the `--character-sets-dir` option when you run the program. For example, to specify a directory to be used by MySQL client programs, list it in the [client] group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

      [client]
      character-sets-dir=/usr/local/mysql/share/mysql/charsets
      [client]
      character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.6/share/charsets"

- If the character set is a complex character set that cannot be loaded dynamically, you must recompile the program with support for the character set.
For Unicode character sets, you can define collations without recompiling by using LDML notation. See Section 1.14.4, “Adding a UCA Collation to a Unicode Character Set”.

- If the character set is a dynamic character set, but you do not have a configuration file for it, you should install the configuration file for the character set from a new MySQL distribution.

- If your character set index file (Index.xml) does not contain the name for the character set, your program displays an error message:

```
Character set 'charset_name' is not a compiled character set and is not specified in the '/usr/share/mysql/charsets/Index.xml' file
```

To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary. However, when `character_set_system` differs from `character_set_server` or `character_set_client`, and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=system_character_set`—that is, setting the client character set to match the system character set—should fix the problem.

### 1.16 MySQL Server Locale Support

The locale indicated by the `lc_time_names` system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()`, and `MONTHNAME()` functions.

`lc_time_names` does not affect the `STR_TO_DATE()` or `GET_FORMAT()` function.

The `lc_time_names` value does not affect the result from `FORMAT()`, but this function takes an optional third parameter that enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable.

Locale names have language and region subtags listed by IANA (http://www.iana.org/assignments/language-subtag-registry) such as 'ja_JP' or 'pt_BR'. The default value is 'en_US' regardless of your system's locale setting, but you can set the value at server startup, or set the `GLOBAL` value at runtime if you have privileges sufficient to set global system variables; see System Variable Privileges. Any client can examine the value of `lc_time_names` or set its `SESSION` value to affect the locale for its own connection.

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)
mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)
mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
| Friday                | January                 |
```
The day or month name for each of the affected functions is converted from utf8 to the character set indicated by the character_set_connection system variable.

lc_time_names may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

<table>
<thead>
<tr>
<th>Locale Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar_AE</td>
<td>Arabic - United Arab Emirates</td>
</tr>
<tr>
<td>ar_BH</td>
<td>Arabic - Bahrain</td>
</tr>
<tr>
<td>ar_DZ</td>
<td>Arabic - Algeria</td>
</tr>
<tr>
<td>ar_EG</td>
<td>Arabic - Egypt</td>
</tr>
<tr>
<td>ar_IN</td>
<td>Arabic - India</td>
</tr>
<tr>
<td>ar_IQ</td>
<td>Arabic - Iraq</td>
</tr>
<tr>
<td>ar_JO</td>
<td>Arabic - Jordan</td>
</tr>
<tr>
<td>ar_KW</td>
<td>Arabic - Kuwait</td>
</tr>
<tr>
<td>ar_LB</td>
<td>Arabic - Lebanon</td>
</tr>
<tr>
<td>ar_LY</td>
<td>Arabic - Libya</td>
</tr>
<tr>
<td>ar_MA</td>
<td>Arabic - Morocco</td>
</tr>
<tr>
<td>ar_OM</td>
<td>Arabic - Oman</td>
</tr>
<tr>
<td>ar_QA</td>
<td>Arabic - Qatar</td>
</tr>
<tr>
<td>ar_SA</td>
<td>Arabic - Saudi Arabia</td>
</tr>
<tr>
<td>ar_SD</td>
<td>Arabic - Sudan</td>
</tr>
<tr>
<td>Locale Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>ar_SY</td>
<td>Arabic - Syria</td>
</tr>
<tr>
<td>ar_TN</td>
<td>Arabic - Tunisia</td>
</tr>
<tr>
<td>ar_YE</td>
<td>Arabic - Yemen</td>
</tr>
<tr>
<td>be_BY</td>
<td>Belarusian - Belarus</td>
</tr>
<tr>
<td>bg_BG</td>
<td>Bulgarian - Bulgaria</td>
</tr>
<tr>
<td>ca_ES</td>
<td>Catalan - Spain</td>
</tr>
<tr>
<td>cs_CZ</td>
<td>Czech - Czech Republic</td>
</tr>
<tr>
<td>da_DK</td>
<td>Danish - Denmark</td>
</tr>
<tr>
<td>de_AT</td>
<td>German - Austria</td>
</tr>
<tr>
<td>de_BE</td>
<td>German - Belgium</td>
</tr>
<tr>
<td>de_CH</td>
<td>German - Switzerland</td>
</tr>
<tr>
<td>de_DE</td>
<td>German - Germany</td>
</tr>
<tr>
<td>de_LU</td>
<td>German - Luxembourg</td>
</tr>
<tr>
<td>el_GR</td>
<td>Greek - Greece</td>
</tr>
<tr>
<td>en_AU</td>
<td>English - Australia</td>
</tr>
<tr>
<td>en_CA</td>
<td>English - Canada</td>
</tr>
<tr>
<td>en_GB</td>
<td>English - United Kingdom</td>
</tr>
<tr>
<td>en_IN</td>
<td>English - India</td>
</tr>
<tr>
<td>en_NZ</td>
<td>English - New Zealand</td>
</tr>
<tr>
<td>en_PH</td>
<td>English - Philippines</td>
</tr>
<tr>
<td>en_US</td>
<td>English - United States</td>
</tr>
<tr>
<td>en_ZA</td>
<td>English - South Africa</td>
</tr>
<tr>
<td>en_ZW</td>
<td>English - Zimbabwe</td>
</tr>
<tr>
<td>es_AR</td>
<td>Spanish - Argentina</td>
</tr>
<tr>
<td>es_BO</td>
<td>Spanish - Bolivia</td>
</tr>
<tr>
<td>es_CL</td>
<td>Spanish - Chile</td>
</tr>
<tr>
<td>es_CO</td>
<td>Spanish - Colombia</td>
</tr>
<tr>
<td>es_CR</td>
<td>Spanish - Costa Rica</td>
</tr>
<tr>
<td>es_DO</td>
<td>Spanish - Dominican Republic</td>
</tr>
<tr>
<td>es_EC</td>
<td>Spanish - Ecuador</td>
</tr>
<tr>
<td>es_ES</td>
<td>Spanish - Spain</td>
</tr>
<tr>
<td>es_GT</td>
<td>Spanish - Guatemala</td>
</tr>
<tr>
<td>es_HN</td>
<td>Spanish - Honduras</td>
</tr>
<tr>
<td>es_MX</td>
<td>Spanish - Mexico</td>
</tr>
<tr>
<td>es_NI</td>
<td>Spanish - Nicaragua</td>
</tr>
<tr>
<td>es_PA</td>
<td>Spanish - Panama</td>
</tr>
<tr>
<td>es_PE</td>
<td>Spanish - Peru</td>
</tr>
<tr>
<td>es_PR</td>
<td>Spanish - Puerto Rico</td>
</tr>
<tr>
<td>Locale Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>es_PY</td>
<td>Spanish - Paraguay</td>
</tr>
<tr>
<td>es_SV</td>
<td>Spanish - El Salvador</td>
</tr>
<tr>
<td>es_US</td>
<td>Spanish - United States</td>
</tr>
<tr>
<td>es_UY</td>
<td>Spanish - Uruguay</td>
</tr>
<tr>
<td>es_VE</td>
<td>Spanish - Venezuela</td>
</tr>
<tr>
<td>et_EE</td>
<td>Estonian - Estonia</td>
</tr>
<tr>
<td>eu_ES</td>
<td>Basque - Spain</td>
</tr>
<tr>
<td>fi_FI</td>
<td>Finnish - Finland</td>
</tr>
<tr>
<td>fo_FO</td>
<td>Faroese - Faroe Islands</td>
</tr>
<tr>
<td>fr_BE</td>
<td>French - Belgium</td>
</tr>
<tr>
<td>fr_CA</td>
<td>French - Canada</td>
</tr>
<tr>
<td>fr_CH</td>
<td>French - Switzerland</td>
</tr>
<tr>
<td>fr_FR</td>
<td>French - France</td>
</tr>
<tr>
<td>fr_LU</td>
<td>French - Luxembourg</td>
</tr>
<tr>
<td>gl_ES</td>
<td>Galician - Spain</td>
</tr>
<tr>
<td>gu_IN</td>
<td>Gujarati - India</td>
</tr>
<tr>
<td>he_IL</td>
<td>Hebrew - Israel</td>
</tr>
<tr>
<td>hi_IN</td>
<td>Hindi - India</td>
</tr>
<tr>
<td>hr_HR</td>
<td>Croatian - Croatia</td>
</tr>
<tr>
<td>hu_HU</td>
<td>Hungarian - Hungary</td>
</tr>
<tr>
<td>id_ID</td>
<td>Indonesian - Indonesia</td>
</tr>
<tr>
<td>is_IS</td>
<td>Icelandic - Iceland</td>
</tr>
<tr>
<td>it_CH</td>
<td>Italian - Switzerland</td>
</tr>
<tr>
<td>it_IT</td>
<td>Italian - Italy</td>
</tr>
<tr>
<td>ja_JP</td>
<td>Japanese - Japan</td>
</tr>
<tr>
<td>ko_KR</td>
<td>Korean - Republic of Korea</td>
</tr>
<tr>
<td>lt_LT</td>
<td>Lithuanian - Lithuania</td>
</tr>
<tr>
<td>lv_LV</td>
<td>Latvian - Latvia</td>
</tr>
<tr>
<td>mk_MK</td>
<td>Macedonian - North Macedonia</td>
</tr>
<tr>
<td>mn_MN</td>
<td>Mongolia - Mongolian</td>
</tr>
<tr>
<td>ms_MY</td>
<td>Malay - Malaysia</td>
</tr>
<tr>
<td>nb_NO</td>
<td>Norwegian(Bokmål) - Norway</td>
</tr>
<tr>
<td>nl_BE</td>
<td>Dutch - Belgium</td>
</tr>
<tr>
<td>nl_NL</td>
<td>Dutch - The Netherlands</td>
</tr>
<tr>
<td>no_NO</td>
<td>Norwegian - Norway</td>
</tr>
<tr>
<td>pl_PL</td>
<td>Polish - Poland</td>
</tr>
<tr>
<td>pt_BR</td>
<td>Portuguese - Brazil</td>
</tr>
<tr>
<td>pt_PT</td>
<td>Portuguese - Portugal</td>
</tr>
<tr>
<td>Locale Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>rm_CH</td>
<td>Romansh - Switzerland</td>
</tr>
<tr>
<td>ro_RO</td>
<td>Romanian - Romania</td>
</tr>
<tr>
<td>ru_RU</td>
<td>Russian - Russia</td>
</tr>
<tr>
<td>ru_UA</td>
<td>Russian - Ukraine</td>
</tr>
<tr>
<td>sk_SK</td>
<td>Slovak - Slovakia</td>
</tr>
<tr>
<td>sl_SI</td>
<td>Slovenian - Slovenia</td>
</tr>
<tr>
<td>sq_AL</td>
<td>Albanian - Albania</td>
</tr>
<tr>
<td>sr_RS</td>
<td>Serbian - Serbia</td>
</tr>
<tr>
<td>sv_FI</td>
<td>Swedish - Finland</td>
</tr>
<tr>
<td>sv_SE</td>
<td>Swedish - Sweden</td>
</tr>
<tr>
<td>ta_IN</td>
<td>Tamil - India</td>
</tr>
<tr>
<td>te_IN</td>
<td>Telugu - India</td>
</tr>
<tr>
<td>th_TH</td>
<td>Thai - Thailand</td>
</tr>
<tr>
<td>tr_TR</td>
<td>Turkish - Turkey</td>
</tr>
<tr>
<td>uk_UA</td>
<td>Ukrainian - Ukraine</td>
</tr>
<tr>
<td>ur_PK</td>
<td>Urdu - Pakistan</td>
</tr>
<tr>
<td>vi_VN</td>
<td>Vietnamese - Vietnam</td>
</tr>
<tr>
<td>zh_CN</td>
<td>Chinese - China</td>
</tr>
<tr>
<td>zh_HK</td>
<td>Chinese - Hong Kong</td>
</tr>
<tr>
<td>zh_TW</td>
<td>Chinese - Taiwan</td>
</tr>
</tbody>
</table>
Chapter 2 MySQL Server Time Zone Support

This section describes the time zone settings maintained by MySQL, how to load the system tables required for named time support, how to stay current with time zone changes, and how to enable leap-second support.

For information about time zone settings in replication setups, see Replication and System Functions and Replication and Time Zones.

• Time Zone Variables
• Populating the Time Zone Tables
• Staying Current with Time Zone Changes
• Time Zone Leap Second Support

Time Zone Variables

MySQL Server maintains several time zone settings:

• The server system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable. The value does not change thereafter.

To explicitly specify the system time zone for MySQL Server at startup, set the `TZ` environment variable before you start `mysqld`. If you start the server using `mysqld_safe`, its `--timezone` option provides another way to set the system time zone. The permissible values for `TZ` and `--timezone` are system dependent. Consult your operating system documentation to see what values are acceptable.

• The server current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial `time_zone` value is 'SYSTEM', which indicates that the server time zone is the same as the system time zone.

Note

If set to `SYSTEM`, every MySQL function call that requires a time zone calculation makes a system library call to determine the current system time zone. This call may be protected by a global mutex, resulting in contention.

The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone` option on the command line, or you can use the following line in an option file:

```
default-time-zone='timezone'
```

If you have the `SUPER` privilege, you can set the global server time zone value at runtime with this statement:

```
SET GLOBAL time_zone = timezone;
```

• Per-session time zones. Each client that connects has its own session time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

```
SET time_zone = timezone;
```

The session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and
Populating the Time Zone Tables

Values for TIMESTAMP columns are converted from the session time zone to UTC for storage, and from UTC to the session time zone for retrieval.

The session time zone setting does not affect values displayed by functions such as UTC_TIMESTAMP() or values in DATE, TIME, or DATETIME columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from TIMESTAMP values. If you want locale-specific arithmetic for DATE, TIME, or DATETIME values, convert them to UTC, perform the arithmetic, and then convert back.

The current global and session time zone values can be retrieved like this:

```sql
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

timezone values can be given in several formats, none of which are case-sensitive:

- As the value 'SYSTEM', indicating that the server time zone is the same as the system time zone.
- As a string indicating an offset from UTC of the form \([H]H:\!MM\), prefixed with a + or -, such as '+10:00', '-6:00', or '+05:30'. A leading zero can optionally be used for hours values less than 10; MySQL prepends a leading zero when storing and retrieving the value in such cases. MySQL converts '-00:00' or '-0:00' to '+00:00'.
- As a named time zone, such as 'Europe/Helsinki', 'US/Eastern', 'MET', or 'UTC'.

Note

Named time zones can be used only if the time zone information tables in the mysql database have been created and populated. Otherwise, use of a named time zone results in an error:

```sql
mysql> SET time_zone = 'UTC';
ERROR 1298 (HY000): Unknown or incorrect time zone: 'UTC'
```

Populating the Time Zone Tables

Several tables in the mysql system database exist to store time zone information (see The mysql System Database). The MySQL installation procedure creates the time zone tables, but does not load them. To do so manually, use the following instructions.

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See Staying Current with Time Zone Changes.

If your system has its own zoneinfo database (the set of files describing time zones), use the mysql_tzinfo_to_sql program to load the time zone tables. Examples of such systems are Linux, macOS, FreeBSD, and Solaris. One likely location for these files is the /usr/share/zoneinfo directory. If your system has no zoneinfo database, you can use a downloadable package, as described later in this section.

To load the time zone tables from the command line, pass the zoneinfo directory path name to mysql_tzinfo_to_sql and send the output into the mysql program. For example:

```sql
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```
Staying Current with Time Zone Changes

The `mysql` command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` system database. Adjust the connection parameters as required.

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or generate leap second information:

- To load a single time zone file `tz_file` that corresponds to a time zone name `tz_name`, invoke `mysql_tzinfo_to_sql` like this:

  ```
  mysql_tzinfo_to_sql tz_file tz_name | mysql -u root -p mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone must account for leap seconds, initialize leap second information like this, where `tz_file` is the name of your time zone file:

  ```
  mysql_tzinfo_to_sql --leap tz_file | mysql -u root -p mysql
  ```

  After running `mysql_tzinfo_to_sql`, restart the server so that it does not continue to use any previously cached time zone data.

If your system has no zoneinfo database (for example, Windows), you can use a package that is available for download at the MySQL Developer Zone:

```
https://dev.mysql.com/downloads/timezones.html
```

**Warning**

Do not use a downloadable time zone package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

You can use either a package that contains SQL statements to populate your existing time zone tables, or a package that contains pre-built MyISAM time zone tables to replace your existing tables:

- To use a time zone package that contains SQL statements, download and unpack it, then load the unpacked file contents into your existing time zone tables:

  ```
  mysql -u root -p mysql < file_name
  ```

  Then restart the server.

- To use a time zone package that contains `.frm`, `.MYD`, and `.MYI` files for the MyISAM time zone tables, download and unpack it. These table files are part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. Stop the server before doing this and restart it afterward.

**Staying Current with Time Zone Changes**

When time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are multiple factors to consider in staying current:
Staying Current with Time Zone Changes

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the website for your operating system vendor for an update that addresses the time changes.

- If you replace the system's `/etc/ localtime` time zone file with a version that uses rules differing from those in effect at `mysqld` startup, restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.

- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date:
  - If your system has its own zoneinfo database, reload the MySQL time zone tables whenever the zoneinfo database is updated.
  - For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace the content of your current time zone tables.

  For instructions for both methods, see Populating the Time Zone Tables. `mysqld` caches time zone information that it looks up, so after updating the time zone tables, restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
```

<table>
<thead>
<tr>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

A count of zero indicates that the table is empty. In this case, no applications currently are using named time zones, and you need not update the tables (unless you want to enable named time zone support). A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, be sure to reload your time zone tables so that applications that use named time zones obtain correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses this query:

```
SELECT
  CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') AS time1,
  CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') AS time2;
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you see an incorrect result like this:

<table>
<thead>
<tr>
<th>time1</th>
<th>time2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-03-11 01:00:00</td>
<td>2007-03-11 02:00:00</td>
</tr>
</tbody>
</table>
After updating the tables, you should see the correct result:

<table>
<thead>
<tr>
<th>time1</th>
<th>time2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-03-11 01:00:00</td>
<td>2007-03-11 01:00:00</td>
</tr>
</tbody>
</table>

Time Zone Leap Second Support

Leap second values are returned with a time part that ends with :59:59. This means that a function such as `NOW()` can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with :59:60 or :59:61 are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with 'YYYY-MM-DD hh:mm:ss' values. The following example demonstrates this. It changes the session time zone to UTC so there is no difference between internal `TIMESTAMP` values (which are in UTC) and displayed values (which have time zone correction applied).

```sql
mysql> CREATE TABLE t1 (a INT, ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP, PRIMARY KEY (ts));
Query OK, 0 rows affected (0.01 sec)
m余个> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)
m余个> -- Simulate NOW() = '2008-12-31 23:59:59'
m余个> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)
m余个> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)
m余个> -- Simulate NOW() = '2008-12-31 23:59:60'
m余个> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)
m余个> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)
m余个> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+------+---------------------+--------------------+
<table>
<thead>
<tr>
<th>a</th>
<th>ts</th>
<th>UNIX_TIMESTAMP(ts)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-12-31 23:59:59</td>
<td>1230767999</td>
</tr>
<tr>
<td>2</td>
<td>2008-12-31 23:59:59</td>
<td>1230768000</td>
</tr>
</tbody>
</table>
+------+---------------------+--------------------+
2 rows in set (0.00 sec)
m余个> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+-----+---------------------+
<table>
<thead>
<tr>
<th>a</th>
<th>ts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2008-12-31 23:59:59</td>
</tr>
</tbody>
</table>
+-----+---------------------+
1 row in set (0.00 sec)
m余个> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in the column, which has the leap second correction applied:
Time Zone Leap Second Support

```sql
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
```

```
+------+---------------------+
<table>
<thead>
<tr>
<th>a</th>
<th>ts</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2008-12-31 23:59:59</td>
</tr>
</tbody>
</table>
+------+---------------------+
1 row in set (0.00 sec)
```