MySQL Quick Reference Card 1.01

http://www.explainth.at

Color key overleaf on Page 4

Introduction

Most websites are hosted on shared servers that already run MySQL. The easiest development option for testing new MySQL databases on Windows XP and Vista is to use a preconfigured Apache for Windows package. We recommend WampServer . This is a pain free way to have a fully functional local Apache + MySQL + PHP installation.

Wamp comes with a web-based MySQL GUI administration module – just startup WampServer, click on the Wamp tray icon and select phpMyAdmin (henceforth called PMA) from the popup menu. This PHP module is offered by most site hosts – either by default or via a simple configuration setting. For local testing an excellent, albeit slightly buggy, alternative to PMA is HeidiSQL.

The user **«root»** used by PMA has no default password assignment. In the interests of security, do the following

- with the MySQL server running execute the command

d:\wamp\bin\mysql\mysql*ver***bin\mysqladmin -u root password** *password*

- With this done edit line 73 of the file **d:\wamp\apps\phpmyadin***ver***config.inc.php** to read

$cfg['Servers'][$i]['password'] = *'password'*;

where **d** is the drive where WampServer was installed and *ver* is the version of the application as installed.

PMA can be used to manipulate MySQL using SQL commands. To do this simply click on the SQL icon on the top l.h.s of the PMA index page. By default the results on issuing SQL in PMA are displayed only partially in a table. Click on ←T→ at the top of the table to view full details.

MySQL Privileges

MySQL identifies users using their username and their location. Consequently, it is perfectly legal to have multiple users with the same username. Hostnames can be a domain name or an IP address. To connect to MySQL from a PHP script running on the server the hostname **localhost** is often – but not always - valid. The correct hostname appears on the top *l.h.s* of the white area of the PMA index/home page.

A user's right to access information in databases on a MySQL is determined by privilege settings stored in the database mysql.

The three most important tables in mysql are user, db and host. Broadly speaking these tables contain columns that help identify the user and others that establish the privileges granted to that user.

User, Host & Password in user establish whether the user should be allowed access to the MySQL server. Privilege data in user is global in scope – i.e. privileges assigned here apply to **all** the databases on the server.

If MySQL finds no privilege information for a user in user it proceeds to examine db. If no match is found in User and Host of db access is denied. A blank User in db represents an anonymous user.

If User is matched and Host is found to be blank MySQL proceeds to examine host. If a row in host contains matching Db and Host the privileges available to the user are established by AND'ing the privilege settings from the relevant rows of db and host.

MySQL allows privilege management at the level of individual stored procedures, tables and even individual columns in tables. The relevant tables are procs_priv, tables_priv and columns_priv. See here for further details.

It should be noted that comparisons with the Host, Column_name and Routine_name columns in the tables discussed above are case insensitive. The columns in these tables are, with some exceptions designed to store 64 characters. Th exceptions – Host (60), User (16) and Password(16).

With the right privileges it is possible to issue SQL(e.g. **INSERT**, **DELETE**) that directly alters the mysql grant tables – e.g. user, db, procs_priv etc. Generally speaking, this not advisable. The **ADD USER**, **DROP USER**, **GRANT** and **REVOKE** statements make a better job of ensuring synchronizing the various tables. However, on occasion it may be necessary to do so to clean up orphaned entries – e.g. tables_priv or columns_priv entries after altering table or column names.

To view the full set of privileges supported by MySQL issue SQL **SHOW PRIVILEGES**;

Charsets & Collations

Charsets are a set of symbols and their encodings. They can be specified at the server, database, table and column level. Some of the more useful MySQL character sets are tabulated below

Charset	Description
latin1	cp1252 West European
latin2	ISO 8859-2 Central European
ascii	US ASCII
utf8	UTF-8 Unicode
cp1250	Windows Central European
cp1251	Windows Cyrillic
cp1256	Windows Baltic
big5	Big5 Traditional Chinese
sjis	Shift-JIS Japanese

SQL: **SHOW CHARACTER SET**; lists available charsets.

Collations are rules for comparing data in a charset. SQL: **SHOW COLLATION**; lists available collations.

Charsets are associated with one or more collations. The default collation for each charset can be viewed by running SQL: **SHOW CHARACTER SET**. Typically collation names consist of the charset followed by a language specification (_lang) and terminating with _ci(case insensitive), _cs(case sensitive) or _bin(binary). As a general rule the charset_general_ci|cs collations make no distinction between related characters – e.g. Á & Â, or s and ß. The _bin collations perform binary comparisons – i.e. they are wholly unaware of concepts such as character case and phonetic similarity. An injudicious choice of collation can have an adverse impact on the speed of SQL execution.

Wamp and most shared Apache site hosts have MySQL set up to use UTF-8 Unicode as the character set with the default collation utf8_unicode_ci.

SQL for creating and altering databases and tables can optionally specify the charset, **X**, and/or collation, **Y**, to be used. This information is used as follows

- With both specified the charset **X** and the collation **Y** are used. Invalid combinations of **X** and **Y** result in an error being reported.
- With **X** specified the default collation for **X** is used.
- With **Y** specified the charset associated with **Y** is used.
- If neither are specified the charset and collation are inherited from the previous level – i.e. the parent server, the database or the table.

Adding/Removing Users

MySQL constrains usernames and passwords to a maximum of 16 characters. The commonly used default MySQL charset of UTF-8 Unicode; shown on the PMA index page; allows the use a wide range of characters in usernames and passwords. User names containing quotation marks, spaces and arithmetic operators **can** be defined by wrapping them in quotes. However, as a general rule, this should be avoided.

Defining New Users

CREATE USER *user*[**@***host*] [**IDENTIFIED BY** *'password'*]

USE mysql;
INSERT INTO user(**Host**,**User**,**Password**)
VALUES(*'host'*,*'user'*,**PASSWORD**(*'password'*));
FLUSH PRIVILEGES;

The user entry created for *user* in user has all privileges disabled – i.e. set to '**N**'. If no host is specified the wildcard % is assumed thereby enabling *user* to access the server from **any** location.

Removing Users

DROP USER *user***@***host;*

USE mysql;
DELETE FROM user WHERE **User** =*'user'* AND **Host** =*'host';*
FLUSH PRIVILEGES;

FLUSH PRIVILEGES instructs MySQL to reload privilege data. Login as a user with adequate privileges in order to issue these SQL commands.

Renaming Users

RENAME USER *user_old*[**@***host_old*] **TO** *user_new*[**@***host_new*] - % is assumed if *host_** is not specified.

Changing Passwords

SET PASSWORD [FOR *user***@***host*] = **PASSWORD(***'password'*);

An error is reported if no such user exists. A user with **UPDATE** privileges on mysql can change his/her own password by omitting the **FOR** clause.

It should be noted that the loopback address, 127.0.0.1,is **not** a synonym for localhost.

Privilege Management

Global Privileges - these settings alter rows in user. Certain privileges, see below, can only be global in scope.

GRANT|REVOKE [*privs***] ON** *.* **TO|FROM** *user*[**@***host*] [**IDENTIFIED BY** *'password'*];

Database Privileges - these settings affect rows in db and may alter rows in user. They apply to all the tables in the database.

GRANT|REVOKE [*privs***] ON** *dbname.** **TO|FROM** *user*[**@***host*] [**IDENTIFIED BY** *'password'*];

Table Privileges – these settings affect rows in tables_priv and may alter rows in user.

GRANT|REVOKE [*privs***] ON** *dbname.tblname* **TO| FROM** *user*[**@***host*] [**IDENTIFIED BY** *'password'*];

Table_name in tables_priv is **NOT UPDATED** when a table is altered.

Table Column Privileges – these settings affect rows in columns_priv and may alter rows in user. Here each privilege must be followed by a parenthesized, comma separated list of column names.

GRANT|REVOKE [*priv***([cols])] ON** *dbname.tblname* **TO|FROM** *user*[**@***host*] [**IDENTIFIED BY** *'password'*];

Column_name and Table_name in columns_priv are **NOT UPDATED** when a table is altered.

Routine Privileges – depending on the nature of the instruction these settings affect rows in user, db or procs_priv.

GRANT|REVOKE CREATE|ALTER ROUTINE ON *.* **TO|FROM** *user*[**@***host*] [**IDENTIFIED BY** *'password'*]; - alters user.

GRANT|REVOKE CREATE|ALTER ROUTINE ON *dbname.** **TO|FROM** *user*[**@***host*] [**IDENTIFIED BY** *'password'*]; - alters db.

GRANT|REVOKE ALTER ROUTINE ON PROCEDURE

MySQL Quick Reference Card 1.01

dbname.procname **TO|FROM** *user*[*@host*] [**IDENTIFIED BY** *'password'*]; - alters procs_priv.

GRANT|REVOKE EXECUTE ON PROCEDURE *dbname.procname* **TO|FROM** *user*[*@host*] [**IDENTIFIED BY** *'password'*]; - alters procs_priv.

MySQL makes no attempt to verify that *procname* actually exists. However dropping *procname* **does** remove the procs_priv rows with matching Routine_name entries.

In all of the above statements, specifying the optional **IDENTIFIED BY** clause results in Password in the matching row of user being altered. **GRANT** statements that specify a non-existent *user*[*@host*] cause a new row to be added to user. However, doing so in a **REVOKE** statement causes MySQL to report that no such grant is defined. Revoking privileges does not remove entries from user. This must be done by issuing a **DROP USER** command.

The creation of a new MySQL user via the **IDENTIFIED** BY clause of the **GRANT** statement can be blocked by using the appropriate sql_mode setting.

The following privileges can only be granted globally – i.e. **ON** *.*

Privilege	Meaning
FILE	Enables use of SELECT ... INTO OUTFILE & LOAD DATA INFILE
PROCESS	Enables use of SHOW PROCESSLIST
RELOAD	Enables use of FLUSH
SHOW DATABASES	Guess?
CREATE USER	Enables use of CREATE USER

The following privileges can be granted at all levels

Privilege	Meaning
ALL	All relevant privileges except GRANT OPTION
ALTER	Enables ALTER TABLE
ALTER ROUTINE	Enables stored routines to be modified or dropped.
CREATE	Enables CREATE TABLE
CREATE ROUTINE	Enables stored routines to be defined
CREATE TEMPORARY TABLE	Guess?
CREATE VIEW	Enables views to be created
DELETE	Allows use of DELETE
DROP	Enables use of DROP TABLE
EXECUTE	Enables user to run stored procedures
INDEX	Enables use of CREATE|DROP INDEX
INSERT	Allows use of INSERT
LOCK TABLES	Enables LOCK TABLES on tables with SELECT privilege.
SELECT	Allows use of SELECT
SHOW VIEW	Enables use of SHOW CREATE VIEW
UPDATE	Allows use of UPDATE
USAGE	NO PRIVILEGES
GRANT OPTION	Allows privileges to be granted

Granting GRANT OPTION privileges requires the special syntax **GRANT... WITH GRANT OPTION**; For instance,

GRANT SELECT ON *.* **TO** *webmaster@localhost* ***WITH GRANT OPTION***;

The GRANT OPTION privilege conferred on a user gives him/her the right to grant **all** his/her privileges to another user. For instance if *webmaster@localhost* has **INSERT** privileges (granted either before or after the above statement was executed) the **GRANT OPTION** would enable him/her to confer both **INSERT and SELECT** privileges on other users. Users with this privilege can club together to boost one another's privileges.

SHOW GRANTS [FOR *user@host*] displays the privileges granted to the user. If the user clause is omitted the privileges for the current user are displayed.

Limiting Connections

GRANT statements can be followed by optional clauses that impose limits on how often the user can connect to the MySQL server.

GRANT...[IDENTIFIED BY *'password'*] **WITH MAX_?** *n*...;

WITH can be followed by one or more of the following.

LIMIT
MAX_QUERIES_PER_HOUR *n*
MAX_UPDATES_PER_HOUR *n*
MAX_CONNECTIONS_PER_HOUR *n*
MAX_USER_CONNECTIONS *n*

Do not use a comma to separate multiple MAX_ settings. For the first three settings a value of 0 implies no limits. For MAX_USER_CONNECTIONS 0 implies that the user should be constrained to the system setting.

It is also possible to impose limits on the nature of the connection.

GRANT...[IDENTIFIED BY *'password'*] **REQUIRE** *require* where

require	Meaning
SSL	Require SSL encrypted connection
X509	Require a valid certificate
CIPHER *'cipher'*	Require SSL with encryption with specified cipher
ISSUER *'issuer'*	Valid certificate from specified issuer.
SUBJECT *'subj'*	Valid certificate with specified subject
NONE	No SSL or certificate requirements

REQUIRE CIPHER implies **SSL**.. Similarly, **REQUIRE ISSUER** and **REQUIRE SUBJECT** imply **X503**. The assignments can be combined. For instance

GRANT...[IDENTIFIED BY *'password'*]

REQUIRE CIPHER *'cipher'* **AND ISSUER** *'issuer'* ;

would require an SSL encrypted connection with a valid certificate emanating from *issuer*. The **REQUIRE** attribute must precede the **WITH MAX_?** attribute if the latter is used.

Data Types

MySQL offers a bewildering range of data types many of which have no equivalent in programming languages such as PHP, Delphi etc.

Integer Types

Type	Bytes	Range (Range Unsigned)
TINYINT	1	-128..127 (0..255)
SMALLINT	2	-32768..32767 (0..65535)
MEDIUMINT	3	-8388608..8388607 (0..16777215)
INT	4	-2147483648..2147483647 (0..4294967295)
BIGINT	8	-9223372036854775808.. 9223372036854775807 (0..18446744073709551615)

INTEGER is a synonym for **INT**.

An unusual, and confusing, feature of MySQL is the ability to specify the display width of integer data types. For instance **INT(*3*)** defines a 4 byte integer with a width of 3. If the value 99 is stored in an **INT**(3) it will be displayed padded to the left with one space character. However, this **does not** alter the intrinsic ability of such an INT column/variable to store integer values far bigger than 999.

Integer column types in **CREATE TABLE** statements and integer local variables in stored procedures can both be given the optional attributes **UNSIGNED** and **ZEROFILL**. Zero-filled columns/variables are automatically unsigned and when displayed use, where relevant, 0 as the padding character.

Floating Point Types

Type	Bytes	Range
FLOAT	4	±1.175494351E-38.. ±3.402823466E+38
DOUBLE	8	±2.2250738585072014E-308.. ±1.7976931348623157E+308

It is possible to define the width and precision of MySQL floating point types by using the syntax **FLOAT|DOUBLE(*m*,*d*)** where *m* is the total number of digits (width) and *d* is the number of digits after the decimal point. The constraint ***m* ≥ *d*** must be respected.

However it is best to avoid such usage – MySQL performs rounding on such column/variable values **before** storage which can lead to unexpected consequences. For instance, storing the value 1.8987 in the column/variable declared as **FLOAT(*3*,*3*)** would actually store 0.999 (the closest possible 3 digit value with a precision of 3).

Floating point types can be given the optional attribute **UNSIGNED**. Negative number assignments to such columns/variables are silently changed to 0.

Both integer and floating point columns can have the additional attribute **AUTO_INCREMENT**. When **NULL** or 0 are inserted into an **AUTO_INCREMENT** column it is automatically assigned the next sequential integer value starting from 1. **LAST_INSERT_ID** reports the most recent **AUTO_INCREMENT** value.

The **DECIMAL** (**NUMERIC**) data type should be used when it is necessary to store floating point values without any roundoff error – e.g. monetary data. This column/variable type is usually specified with a width an precision specifier, e.g. **DECIMAL(*m*,*d*)** with ***m* ≥ *d***. Here too, unexpected rounding can occur. For instance, storing 100.0097 in a column/variable declared as **DECIMAL(*3*,*2*)** would store 9.99 – the closest value with 2 decimal digits and 3 digits in total.

The **BIT(*M*)**, *M* = 1..64, datatype provides storage for bitfields. Assignments to bit columns/variables can be made as integers or using the format **b***'ddd'* where *d* is 0 or 1. Strings *'ddd'* is shorter than the bitfield length are left padded with 0s. This datatype requires (*M* + 7)/8 bytes (rounded up) of storage.

Date & Time Types

Type	Bytes	Description
DATE	3	Date as YYYY-MM-DD
DATETIME	8	Date & Time as YYYY-MM-DD HH:MM:SS
TIMESTAMP	4	Date & Time as YYYY-MM-DD HH:MM:SS
TIME	3	Time as HH:MM:SS
YEAR[(*2*|*4*)]	1	Year as YY or YYYY (Default)

The valid ranges for each of these data types are tabulated below

MySQL Quick Reference Card 1.01

Type	Range
DATE	1000-01-01 to 9999-12-31
DATETIME	1000-01-01 00:00:00 to 9999-12-31 23:59:59
TIMESTAMP	1970-01-01 00:00:01 **UTC** to 2038-01-09 03:14:07 **UTC**
TIME	-838:59:59 to 838:59:59
YEAR	YYYY:1901 to 2155 YY:70(1970) to 69(2069)

Assigning 0 or '0' to any of these data types yields the following

Type	Contents
DATE	0000-00-00
DATETIME/TIMESTAMP	0000-00-00 00:00:00
TIME	00:00:00
YEAR	0000

Out of range **TIME** assignment result in the closest possible (±838:59:59) value being used. With all other date/time types out of range assignments get converted to the corresponding zero values. Zero assignments trigger a warning If sql_mode contains **NO_ZERO_DATE**.

Assignments to these datatypes can be made in a number of different formats

Example	Result
DATE	
'70111'[1]	1970-11-01
'070609'	2007-06-09
17760404	1776-04-04
'2001.09.11'	2001-09-11
19450806	1945-08-06
'1945@08.08'	1945-08-08
DATETIME & TIMESTAMP[2]	
1945080**8081500**	1945-08-08 08:15:00
'1945.08/08 **08+15*00**'	1945-08-08 08:15:00
20070609143100	2007-06-09 14:31:00
810511233100	1981-05-11 23:31:00
TIME[3]	
'1 02:02:02'	26:02:02
'01:30:45'	01:30:45
'2 06:05'	54:05:00
'3 01'	73:00:00
16:15	16:15:00
43	00:00:43
YEAR[4]	
23	2023
79	1979
'1901'	1901
0	0000
'00'	2000

[1] Not recommended.

[2] Time part shown in bold

[3] **TIME** can be used to store time differences, not just the time of the day. The most generic format is '**D HH:MM:SS**' 0 ≤ **D** ≤ 34. Some superior or inferior parts can be left out if the resulting string makes temporal sense.

[4] Two digit year assignments, both string and integer, are interpreted in a special way. '00' to '69' are treated as additions to the year 2000. '70' to '99' are treated as years from 1970 to 1999. A 2 digit assignment for the year 2000 can only be made as a string.

String Types

The **CHARACTER SET** and **COLLATION** attributes are often specified with string columns/variables. Failing this, the string inherits this information from a prior level – table, database or server. The storage requirements for strings depend both on the precise data type and the **CHARACTER SET** used.

Type	Width (Bytes)
CHAR(*M*), 0 ≤ *M* ≤ 255	*M* x *w*[5]
VARCHAR(*M*), 0 ≤ *M* ≤ 65535	𝕃 + 1|2[6,7]
TEXT[(*M*)][8]	𝕃 + 2
MEDIUMTEXT	𝕃 + 3
LONGTEXT	𝕃 + 4

[5] *w* is maximum bytes required by the charset.

[6] 𝕃 is actual byte length of the string. One additional byte is required only if ***M* ≤ 255.**

[7] MySQL imposes a maximum length of 65535 on table rows. **VARCHAR** fields have to share this space.

[8] If M is specified, MySQL uses the smallest datatype that can hold the string.

TEXT, **MEDIUMTEXT** and **LONGTEXT** types can store a maximum of **$2^n - 1$** characters (**n** = 16|24|32). The actual number depends on the charset used.

Enumerations

ENUM(*'val1','val2'..*)**NULL**) **[CHARACTER SET** *charset***]** **[COLLATION** *collation***]**

MySQL enumerations are strings which can be assigned one of the specified values. Internally, they are stored as integers. An enumeration can have up to 65535 distinct values.

Sets

SET(*'val1','val2'..*) **[CHARACTER SET** *charset***]** **[COLLATION** *collation***]**

MySQL sets can be assigned zero or more values from the range specified. Internally they are stored as an integer.

Blobs

BLOB stands for Binary Large Object. In, MySQL there is a matching **BLOB** type for each **TEXT** type – e.g. **LONGBLOB** - with precisely the same storage requirements However, no charset or collation information is associated with blobs. Sorting and comparison of blobs is performed based simply on the value of the byte sequences they store.

The assignment behavior of the data types discussed here depends on the sql_mode setting.

SQL_MODE

sql_mode determines the SQL syntax MySQL supports and the data validation it performs. MySQL understands two kinds of modes – global and session. Both settings are in effect a set of options. The current settings can be viewed by issuing SQL

SELECT @@[session|global].sql_mode;

To assign sql_mode issue SQL

SET [session|global] sql_mode= [*modes*];

The session setting affects only the current user. However, it would be incorrect to assume that setting the session level mode will ensure use of the specified data validation rules at all times throughout the duration of the session. Stored procedures and triggers use the mode that was in effect *at the time they were **defined***. This information is stored in

information_schema.routines.sql_mode and **information_schema.triggers.sql_mode**

The *modes* assignment above takes the form of a string containing a comma separated list of one or more of the following options

- ALLOW_INVALID_DATES – Constrains date/time data type checking to valid month and day numbers. Invalid dates, e.g. 2008-04-31 are accepted.
- ANSI_QUOTES – Treat the double quote, ", as an identifier quote character – i.e. used to quote identifiers containing special characters or SQL keywords. The default identifier quote ' (**ALT** + 96) can always be used
- ERROR_FOR_DIVISION_BY_ZERO – trigger error rather than warning for such errors in **INSERT** and **UPDATE** operations. If the **IGNORE** clause is specified a warning is generated.
- HIGH_NOT_PRECEDENCE – Gives **NOT** a higher precedence. With this setting **NOT *1* BETWEEN *-5* AND *5*** is treated as **NOT(*1*) BETWEEN *-5* AND *5***
- IGNORE_SPACE – allow spaces between function name and **(**. With this setting identifiers that are SQL functions must be quoted - using backticks ` or double quotes if ANSI_QUOTES is set.
- NO_AUTO_CREATE_USER – Limits **GRANT** statements to changing user privileges.
- NO_AUTO_VALUE_ON_ZERO – Blocks 0 entries into **AUTO_INCREMENT** columns being converted into next sequential auto value.
- NO_BACKSLASH_ESCAPES – \ in strings is not treated as escape sequence indicator.
- NO_DIR_IN_CREATE – Ignore **INDEX|DATA DIRECTORY** in **CREATE TABLE** statements.
- NO_ENGINE_SUBSTITUTION – prevents use of the default storage engine if the specified one is not available.
- NO_FIELD|KEY|TABLE_OPTIONS – MySQL specific options not displayed in **SHOW CREATE TABLE** output.
- NO_UNSIGNED_SUBTRACTION – subtraction result is always signed.
- NO_ZERO_DATE – 0000-00-00 is not a valid date. Can be overridden locally by using **IGNORE**.
- NO_ZERO_IN_DATE – Block date entries where day/month parts are zero. When used with **IGNORE** the value is converted to a zero date.
- ONLY_FULL_GROUP_BY – All **SELECT** columns must be specified in **GROUP BY** clause.
- PIPES_AS_CONCAT - || is treated as a synonym for **CONCAT**.
- REAL_AS_FLOAT – treats **REAL** as synonym for **FLOAT** not **DOUBLE**.
- STRICT_ALL_TABLES – Strict data validation for all tables. **INSERT** and **UPDATE** operations are abandoned as soon as an error is encountered. This can result in partial updates.
- STRICT_TRANS_TABLES – Strict data validation for tables using transactional storage. Invalid values are adjusted. Missing values are replaced with the default for the column type. In both cases MySQL issues a warning and continues.

Strict modes block invalid dates, e.g. **1987-02-29** but allow zero dates and zero values in the date/month parts. The NO_ZERO_? modes should be included to prevent this. The effects of strict modes can be overridden locally by using **INSERT|UPDATE IGNORE**.

MySQL provides short hand notation for specifying some of the more commonly used mode combinations.

- **ANSI =** REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE
- **DB2 =** PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS
- **MSSQL =** PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS
- **ORACLE =** PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER
- **POSTGRESSQL =** PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS
- **TRADITIONAL =** STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE,

MySQL Quick Reference Card 1.01

ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER

Storage Engines

Storage engine refers to the software layer MySQL handles **INSERT**, **SELECT**, **UPDATE** and **DELETE** operations[1].

Engine	Description
MyISAM	Non-transactional (**NT**), high speed with fulltext search capabilities
Memory	**NT** engine for in-memory tables
MERGE	**NT** engine to handle identical MyISAM tables as a single table
InnoDB	Storage engine for transaction safe tables
EXAMPLE	A stub engine with no data storage/retrieval capabilities
FEDERATED	Engine for **CRUD** operations on remote tables – i.e. stored in remote databases, not local tables
ARCHIVE	Used for space efficient storage of large amounts of data without indexing
BLACKHOLE	A "no-op" engine useful for verifying dump file syntax and identifying bottlenecks not related to the storage engine
CSV	For data storage in CSV format text files. Data are stored in tablename.csv. Indexing is not supported

MySQL databases may contain tables that use different storage engines. Transaction safe tables offer the ability to have a number of **CRUD** operations executed at the same time. The changes can be rolled back if necessary. Their contents are easier to retrieve in the event of software or hardware failure. **NT** tables offer faster access, consume lesser disk space and have lower memory requirements during updates.

[1] These are the SQL equivalents of the classical persistent storage operations of Create, Retrieve, Update and Delete (**CRUD**)

SHOW Statements

MySQL supports an extensive range of **SHOW** statements that provide useful information for database administrators.

Statement[1]	Shows
CHARACTER SET[2]	Available charsets
COLLATION[2]	Available collations
[FULL] COLUMNS FROM *tblname* **[FROM** *dbname***]**	Column information. **FULL** displays comments, collation and current user privileges for each column.
CREATE DATABASE *dbname*	SQL required to create *dbname*
CREATE PROCEDURE| FUNCTION *dbname.procname*	SQL required to create *procname*.
CREATE VIEW *viewname*	SQL required to create *viewname*
DATABASES[2]	databases on server. User must have some privileges on the database or have the global **SHOW DATABASES** privilege.
ENGINE *engine* **LOGS| STATUS**	Information on *engine*. **LOGS** is not supported by all versions.
ENGINES	Engine name, comment and support information. The support column reads YES, NO, DEFAULT or DISABLED.
ERRORS [LIMIT [*offset***,]***rows***]**	Errors from last SQL that generated messages. Optionally, *rows* errors optionally starting from *offset'th* message.
GRANTS [*user***]**	Grants current user if no *user* specified.
INDEX FROM *tblname* **[FROM** *dbname***]**	Index information
OPEN TABLES [FROM *dbname***]**[2]	Tables currently open in the table cache. Optionally, only those in *dbname*
PROCEDURE| FUNCTION CODE *procname*	Internal implementation of *procname*. Only available if server was built with debugging support.
PROCEDURE| FUNCTION STATUS[2]	Routine information – database, name etc.
[SESSION|GLOBAL] STATUS[2]	Server status
TABLE STATUS [FROM *dbname***]**[2]	Detailed table information from *dbname* or current database.
TABLES [FROM *dbname***]**[2]	Lists tables in *dbname* or current database.
TRIGGERS [FROM *dbname***]**[2]	Triggers defined for tables in *dbname* or current database.
[SESSION|GLOBAL] VARIABLES[2]	Session or global MySQL system variables.
WARNINGS [LIMIT [*offset***,]***rows***]**	Like **SHOW ERRORS** but displays warnings, notes and errors.

[1] All statements should begin with **SHOW**
[2] Optionally with **LIKE** *'pattern'* or **WHERE** *expr*.

Statements that allow *dbname* may report an error if no database has been selected and *dbname* is not specified.. Issue a **USE** *dbname* to avoid this.

Color Key

PMA - Identifier for phpMyAdmin
SQL - Structured Query Language
ver - Server installation dependent placeholder text
text - Placeholder text
$cfg - PHP variable.
'password' - PHP array index
database - Database name
table - Table name
Column - Column name
INSERT – SQL keyword
A|B – A or B. A is the default.
A|B...C|D – Paired options. A and C or B and D.
[...] - Optional clause
m – number in SQL statement.
{...} – set of options. e,g, red,blue..green.
sql_mode – global MySQL variable
IGNORE_SPACE – set element.
«root» MySQL user