WebsiteSetup

# MySQL Cheat Sheet

Ready to advance your coding skills and master databases? Great! Then you will find our MySQL cheat sheet absolutely handy.

MySQL is a popular, open-source, relational database that you can use to build all sorts of web databases — from simple ones, cataloging some basic information like book recommendations to more complex data warehouses, hosting hundreds of thousands of records. Learning MySQL is a great next step for those who already know PHP or Perl. In this case, you can create websites that interact with a MySQL database in real-time and display searchable and categorized records to users.

**Sounds promising? Let's jump in then!**

# Table of Contents

## MySQL 101: Getting Started

Similar to other programming languages like PHP, JavaScript, HTML, and jQuery, MySQL relies on commenting to execute any commands.

You can write two types of comments in MySQL:

- **Single-Line Comments**: These start with "–". Any text that goes after the dash and till the end of the line will not be taken into account by the compiler.

**Example:**

```
–Update all:
SELECT * FROM Movies;
```

- **Multi-Line Comments:** These start with /* and end with */. Again, any text that is beyond the slashes lines will be ignored by the compiler.

**Example:**

```
/*Select all the columns
of all the records
in the Movies table:*/
SELECT * FROM Movies;
```

Keeping this in mind, let's get started with actual coding.

## How to Connect to MySQL

To start working with MySQL, you'll need to establish an active SSH session on your server.

```
mysql –u root –p
```

If you didn't set a password for your MySQL root user, you omit the -p switch.

## Create a new MySQL User Account

Next, you can create a new test user for practice. To do that, run the following command:

```
CREATE USER 'username'@'localhost' IDENTIFIED BY 'password';
```

If you need to delete a user later on you, use this command:

```
DROP USER 'someuser'@'localhost';
```

## Create a New Database

To set up a new database use this line:

```
CREATE DATABASE yourcoolname
```

You can then view all your databases with this command:

```
mysql> show databases;
```

Later on, you can quickly navigate to a particular database using this command:

```
[root@server ~]# mysql -u root -p mydatabase < radius.sql
```

## Delete a MySQL Database

To get rid of a database just type:

```
DROP DATABASE dbName
```

If you are done for the day, just type "exit" in the command line to finish your session.

## Essential MySQL Commands

**SELECT** — choose specific data from your database
**UPDATE** — update data in your database
**DELETE** — deletes data from your database
**INSERT INTO** — inserts new data into a database
**CREATE DATABASE** — generate a new database
**ALTER DATABASE** — modify an existing database
**CREATE TABLE** — create a new table in a database
**ALTER TABLE** — change the selected table
**DROP TABLE** — delete a table
**CREATE INDEX** — create an index (search key for all the info stored)
**DROP INDEX** — delete an index

# Working with Tables

Tables are the key element of MySQL databases as they let you store all the information together in organized rows. Each row consists of columns that feature a specified data type. You have plenty of options for customization using the commands below.

## Create a New Simple Table

Use this command to create a new table:

```
CREATE TABLE [IF NOT EXISTS] table_name(
  column_list
);
```

The code snippet below features a table for a list of movies that we want to organize by different attributes:

```
CREATE TABLE movies(
   title VARCHAR(100),
   year VARCHAR(100),
   director VARCHAR(50),
   genre VARCHAR(20),
   rating VARCHAR(100),
);
```

## View Tables

Use the next commands to get more information about the tables stored in your database.

**show tables** — call a list of all tables associated with a database.

**DESCRIBE table_name;** — see the columns of your table.

**DESCRIBE table_name column_name;** — review the information of the column in your table.

## Delete a Table

To get rid of the table specify the table name in the following command:

```
DROP TABLE tablename;
```

# Working With Table Columns

Use columns to store alike information that shares the same attribute (e.g. movie director names). Columns are defined by different storage types:

- **CHAR**
- **VARCHAR**
- **TEXT**
- **BLOB**
- **EUT**
- And others.

An in-depth overview comes in the next section!

When designing columns for your database, your goal is to select the optimal length to avoid wasted space and maximize performance.

Below are the key commands for working with tables.

## Add New Column

```
ALTER TABLE table
ADD [COLUMN] column_name;
```

## Delete/Drop a Column

```
ALTER TABLE table_name
DROP [COLUMN] column_name;
```

## Insert New Row

```
INSERT INTO table_name (field1, field2, ...) VALUES (value1,
value2, ...)
```

## Select Data from The Row

Specify what kind of information you want to retrieve from a certain row.

```
SELECT value1, value2 FROM field1
```

## Add an Additional Selection Clause

Include an additional pointer that indicates what type of data do you need.

```
SELECT * FROM movies WHERE budget='1';
SELECT * FROM movies WHERE year='2020' AND rating='9';
```

## Delete a Row

Use SELECT FROM syntax and WHERE clause to specify what rows to delete.

```
DELETE FROM movies WHERE budget='1';
```

## Update Rows

Similarly, you can use different clauses to update all or specified rows in your table.
To update all rows:

```
UPDATE table_name
SET column1 = value1,
    ...;
```

To update data only in a specified set of rows you can use WHERE clause:

```
UPDATE table_name
SET column_1 = value_1,
WHERE budget='5'
```

You can also update, select or delete rows using JOIN clause. It comes particularly handy when you need to manipulate data from multiple tables in a single query.
Here's how to update rows with JOIN:

```
UPDATE table_name
INNER JOIN table1 ON table1.column1 = table2.column2
SET column1 = value1,
WHERE budget='5'
```

## Edit a Column

You can alter any existing column with the following snippet:

```
ALTER TABLE movies MODIFY COLUMN  number INT(3)
```

## Sort Entries in a Column

You can sort the data in all columns and rows the same way you do in Excel e.g. alphabetically or from ascending to descending value.

```
SELECT * FROM users ORDER BY last_name ASC;
SELECT * FROM users ORDER BY last_name DESC;
```

## Search Columns

Here's how you can quickly find the information you need using WHERE and LIKE syntax:

```
SELECT * FROM movies WHERE genre LIKE 'com%';
SELECT * FROM movies WHERE title LIKE '%a';
```

You can also exclude certain items from search with NOT LIKE:

```
SELECT * FROM movies WHERE genre NOT LIKE 'hor%';
```

## Select a Range

Or you can bring up a certain data range using the next command:

```
SELECT * FROM movies WHERE rating BETWEEN 8 AND 10;
```

## Concentrate Columns

You can mash-up two or more columns together with CONCAT function:

```
SELECT CONCAT(first_name, ' ', last_name) AS 'Name', dept FROM
users;
```

# Data Types

Data types indicate what type of information you can store in a particular column of your table. MySQL has three main categories of data types:

- Numeric
- Text
- Date/time

## Numeric Data Types

Unless programmed, the MySQL column display width will not limit the range of values that you can store there. Also, without a numeric data type integer, your columns can display width incorrectly if you include too wide values. To prevent that you can use the following integers to specify the maximum allowed range of values. You can either:

- Assign a specific numeric value to the column
- Or leave an **unsigned** value.

If unsigned, the column will expand to hold the data up till a certain upper boundary range.

- **BIT[(M)]** — specify a bit-value type. **M** stands for the number of bits per value, ranging from 1 to 64. The default is 1 if no T specified.
- **ZEROFILL** — auto-add UNSIGNED attribute to the column. Deprecated since the MySQL 8.0.17 version.
- **TINYINT(M)** — the smallest integer with a range of  -128 to 127.
  - **TINYINT(M) [UNSIGNED]** — the range is 0 to 255.
  - **BOOL, BOOLEAN** — synonyms for TINYINT(1)
- **SMALLINT(M)** — small integer with a range of -32768 and 32767.
  - **SMALLINT(M) [UNSIGNED]** — the range is 0 to 65535.
- **MEDIUMINT(M)** — medium integer with a range of -8388608 to 8388607.
  - **MEDIUMINT(M) [UNSIGNED]** — the range is 0 to 16777215.
- **INT(M) and INTEGER (M)** — normal range integer with a range of -2147483648 to 2147483647.
  - **INT(M)[UNSIGNED] and INTEGER (M)[UNSIGNED]** — the range is 0 to 4294967295.
- **BIGINT(M)** — the largest integer with a range of -9223372036854775808 to 9223372036854775807.
  - **BIGINT(M) [UNSIGNED]** — the range is 0 to 8446744073709551615.
- **DECIMAL (M, D)** — store a double value as a string. **M** specifies the total number of digits. **D** stands for the number of digits after the decimal point. Handy for storing currency values.
  - Max number of M is 65. If omitted, the default M value is 10.
  - Max number of D is 30. If omitted, the default D is 0.
- **FLOAT (M, D)** — record an approximate number with a floating decimal point. The support for FLOAT is removed as of MySQL 8.0.17 and above.
  - Permissible values ranges are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.

## Blob and Text Data Types

**BLOB** binary range enables you to store larger amounts of text data. The maximum length of a BLOB is **65,535 ($2^{16}$ – 1)** bytes. BLOB values are stored using a 2-byte length prefix.

**NB:** Since text data can get long, always double-check that you do not exceed the maximum lengths. The system will typically generate a warning if you go beyond the limit. But if nonspace characters get truncated, you may just receive an error without a warning.

- **TINYBLOB** — sets the maximum column length at 255 ($2^8$ – 1) bytes. TINYBLOB values are stored using a 1-byte length prefix.
- **MEDIUMBLOB** — sets the maximum column length at 16,777,215 ($2^{24}$ – 1) bytes. MEDIUMBLOB values are stored using a 3-byte length prefix.
- **LONGBLOB** — sets the maximum column length at 4,294,967,295 or 4GB ($2^{32}$ – 1) bytes. LONGBLOB values are stored using a 4-byte length prefix.

  **Note**: The max length will also depend on the maximum packet size that you configure in the client/server protocol, plus available memory.

**TEXT** does the same job but holds values of smaller length. A TEXT column can have a maximum length of **65,535 ($2^{16}$ – 1) characters**. However, the max length can be smaller if the value contains multibyte characters. TEXT value is also stored using a 2-byte length prefix.

- **TINYTEXT** — store a value using a 1-byte length prefix. The maximum supported column length is 255 ($2^8$ – 1) characters.
- **MEDIUMTEXT** — store a value using a 3-byte length prefix. The maximum supported column length is 16,777,215 ($2^{24}$ – 1) characters.
- **LONGTEXT** — store a value using a 4-byte length prefix. The maximum supported column length is 4,294,967,295 or 4GB ($2^{32}$ – 1) characters.

  **Note:** Again, the length cap will also depend on your configured maximum packet size in the client/server protocol and available memory.

## Text Storage Formats

- **CHAR** — specifies the max number of non-binary characters you can store. The range is from 0 to 255.
- **VARCHAR** — store variable-length non-binary strings. The maximum number of characters you can store is 65,535 (equal to the max row size).
  - VARCHAR values are stored as a 1-byte or 2-byte length prefix plus data, unlike CHAR values.
- **BYNARY** — store binary data in the form of byte strings. Similar to CHAR.
- **VARBYNARY** — store binary data of variable length in the form of byte strings. Similar to VARCHAR.
- **ENUM** — store permitted text values that you enumerated in the column specification when creating a table.
  - ENUM columns can contain a maximum of 65,535 distinct elements and have > 255 unique element list definitions among its ENUM.
- **SET** — another way to store several text values that were chosen from a predefined list of values.
  - SET column can contain a maximum of 64 distinct members and have > 255 unique element list definitions among its SET.

## Date and Time Data Types

As the name implies, this data type lets you store the time data in different formats.

- **DATE** — use it for values with a date part only. MySQL displays DATE values in the 'YYYY-MM-DD' format.
  - Supported data range is '1000-01-01' to '9999-12-31'.

- **DATETIME** — record values that have both date and time parts. The display format is 'YYYY-MM-DD hh:mm:ss'.
  - Supported data range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

- **TIMESTAMP** — add more precision to record values that have both date and time parts, up till microseconds in UTC.
  - Supported data range is '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.

- **TIME** — record just time values in either 'hh:mm:ss' or 'hhh:mm:ss' format. The latter can represent elapsed time and time intervals.
  - Supported data range is '-838:59:59' to '838:59:59'.

- **YEAR** — use this 1-byte type used to store year values.
  - A 4-digit format displays YEAR values as 0000, with a range between 1901 to 2155.
  - A 2-digit format displays YEAR values as 00. The accepted range is '0' to '99' and MySQL will convert YEAR values in the ranges 2000 to 2069 and 1970 to 1999.

# Working With Indexes

**Indexes** are the core element of your database navigation. Use them to map the different types of data in your database, so that you don't need to parse all the records to find a match.

**NB**: You have to update an index every time you are creating, changing or deleting a record in the table. Thus, it's best to create indexes only when you need to and for frequently searched columns.

## How to Create an Index

The basic syntax is as follows:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

You can also create a unique index — one that enforces the uniqueness of values in one or more columns.

```
CREATE UNIQUE INDEX index_name
ON table_name(index_column_1,index_column_2,...);
```

## How to Delete an Index in MySQL

Use the DROP command for that:

```
DROP INDEX index_name;
```

# Working with Views

A **view** is a virtual representation of an actual table that you can assemble up to your liking (before adding the actual one to your database).

It features rows and columns, just like the real deal and can contain fields from one or more of the real tables from your database. In short, it's a good way to visualize and review data coming from different tables within a single screen.

## How to Create a New View

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

## Update a View

A view always displays fresh data since the database engine recreates it each time, using the view's SQL statement. To refresh your view use the next code:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

## Rename a View

If you are dealing with multiple views at a time, it's best to give them distinctive names. Here's how that done:

```
RENAME TABLE view_name TO new_view_name;
```

## Show All Views

To call up all current views for all tables from the database, use this snippet:

```
SHOW FULL TABLES
WHERE table_type = ‘VIEW’;
```

## Delete a View

To delete a single view use the DROP command:

```
DROP VIEW [IF EXISTS] view_name;
```

You can also delete multiple views at a time:

```
Drop Multiple views: DROP VIEW [IF EXISTS] view1, view2, ...;
```

# Working With Triggers

A **trigger** is a database object, associated with a table. It activates whenever a specific event happens for the table.

For example, you can set up triggers for events such as:

- Row or deletes updates
- Row information inserts

This is a more advanced topic, so check the official MySQL trigger FAQ section for more details.

## How to Create a Trigger

To create a simple trigger that will pop up before or after a certain operation such as INSERT, UPDATE or DELETE, use this code:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
ON table_name FOR EACH ROW
trigger_body;
```

## Review All Triggers in Your Database

Search your database for all the active triggers using LIKE and WHERE clauses.

```
SHOW TRIGGERS
[{FROM | IN} database_name]
[LIKE ‘pattern’ | WHERE search_condition];
```

## How to Delete a Trigger

To remove a trigger, use the DROP command:

```
DROP TRIGGER [IF EXISTS] trigger_name;
```

# Stored Procedures for MySQL

**Stored procedures** are reusable SQL code snippets that you can store in your database and use-as-needed over and over again. They save you tons of time since you don't need to write a query from scratch. Instead, you just call it to execute it.

## How to Create a Stored Procedure in MySQL

Here's how to create a simple stored procedure with no additional parameters:

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

And here's another stored procedure example featuring WHERE clause:

```
CREATE PROCEDURE SelectAllMovies @Title varchar(30)
AS
SELECT * FROM Movies WHERE Title = @Title
GO;
```

## Review All Stored Procedures

Similarly to triggers, you can review all stored procedures with LIKE and WHERE:

```
SHOW PROCEDURE STATUS
[LIKE 'pattern' | WHERE search_condition];
```

## How to Delete a Stored Procedure

To get rid of a stored procedure you no longer need, use DROP:

```
DROP PROCEDURE [IF EXISTS] procedure_name;
```

# Logical Operators

**Logical operators** enable you to add more than one condition in WHERE clause. This makes them super handy for more advanced search, update, insert and delete queries.

In MySQL you have three main logical operators:

- **AND** — use it to filter records that rely on 1+ condition. This way you can call records that satisfy all the conditions separated by AND.
- **OR** — call records that meet any of the conditions separated by OR.
- **NOT** — review records that do not meet a certain condition (e.g. NOT blue). It's a handy operator from excluding certain data.

Plus, some additional special operators:

- **BETWEEN** — select or search data between a range of set min and max values.
- **LIKE** — compare one record to another. Handy operator for search.
- **IS NULL** — compare some value with a NULL value.
- **IN** — determine if a value or expression matches one of the values on your list.
- **ALL** — compare a value or expression to all other values in a list.
- **ANY** — compare a value or expression to any value in your list according to the specified condition.
- **EXISTS** — test if a certain record exists.

# Aggregate Functions

**Aggregate functions** in MySQL allow you to run a calculation on a set of values and return a single scalar value. In essence, they are a great way to find the needed data faster and organize it better using **GROUP BY** and **HAVING** clauses of the **SELECT** statement.

Below is an overview of these:

## MIN

Find the smallest value of the selected column in your table:

```
SELECT MIN (column_name)
FROM table_name
WHERE condition;
```

## MAX

Does the opposite and returns the largest value of the selected column:

```
SELECT MAX (column_name)
FROM  table_name
WHERE condition;
```

## COUNT

Call up several rows that meet the specified criteria:

```
SELECT COUNT (column_name)
FROM table_name
WHERE condition;
```

## AVG

Get the average value of a numeric column that you selected:

```
SELECT AVG (column_name)
FROM table_name
WHERE condition;
```

## SUM

Receive a total sum of a numeric column that you selected:

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

## Arithmetic, Bitwise, Comparison, and Compound Operators

| Arithmetic Operators | Bitwise Operators | Comparison Operators | Compound Operators |
|---|---|---|---|
| `+, -, *, /, %` | `&, \|, ^` | `= =, <, >, <=, >=, < >` | `+=, *=, -=, /=, %=, &=, ^-+, \| *=` |

Source: Edureka

## SQL Database Backup Commands

Finally, don't forget to regularly backup your progress as you are testing different commands and code snippets.

There are several easy ways to do it. To backup your database to SQL file, use this code:

```
mysqldump -u Username -p dbNameYouWant > databasename_backup.sql
```

Then, to restore your work from a SQL backup, run the following line:

```
mysql - u Username -p dbNameYouWant < databasename_backup.sql
```

## Conclusions

Learning how to code MySQL databases may seem like a tedious task at first. But once you master the basic MySQL commands and syntax, you are set for success. Knowing MySQL can give you an edge in web development, especially with e-commerce websites and online stores.

The MySQL cheat sheet above is great for beginners. Grab your JPEG copy and bookmark this page for quick access!

*If you have any questions or want to add something to our MySQL checklist, leave a quick comment below!*

WebsiteSetup

MySQL®

**REFERENCE SHEET**

versions 3.23, 4.0, 4.1

## numeric

| | |
|---|---|
| **TINYINT***[(digits)] [unsigned\|zerofill]* | *256* |
| **BIT,BOOL,BOOLEAN** | *synonyms for tinyint(1)* |
| **SMALLINT***[(digits)] [unsigned\|zerofill]* | *65,536* |
| **MEDIUMINT***[(digits)] [unsigned\|zerofill]* | *16,777,216* |
| **INT,INTEGER***[(digits)] [unsigned\|zerofill]* | *4,294,967,296* |
| **BIGINT***[(digits)] [unsigned\|zerofill]* | *18,446,744,073,709,551,616* |
| **FLOAT***[(digits, digits after decimal)] [unsigned\|zerofill]* | *23 digits* |
| **DOUBLE***[(digits, digits after decimal)] [unsigned\|zerofill]* | *24…53 digits* |
| **DECIMAL***[(digits, digits after decimal)] [unsigned\|zerofill]* | *a type of DOUBLE stored as a string* |

### functions

| | |
|---|---|
| **ABS(***X***)** | **SIGN(***X***)** |
| **FLOOR(***X***)** | **CEILING(***X***)** |
| **ROUND(***X[,D]***)** | **EXP(***X***)** |
| **DIV(***X***)** | **MOD(***N,M***)** |
| **POW(***X,Y***)** | **POWER(***X,Y***)** |
| **SQRT(***X***)** | **RAND(***[seed]***)** |
| **PI()** | **DEGREES(***X***)** |
| **RADIANS(***X***)** | **COT(***X***)** |
| **COS(***X***)** | **ACOS(***X***)** |
| **SIN(***X***)** | **ASIN(***X***)** |
| **TAN(***X***)** **ATAN(***X***)** | **ATAN2(***X***)** |

**LOG(***X***), LOG2(***X***), LOG10(***X***)** **LN(***X***)**
**TRUNCATE(***X, D***)**

## strings

| | |
|---|---|
| **CHAR***[(length)]* | *0…255 – fixed length, right-padded with spaces* |
| **VARCHAR***[(length)]* | *0…255 – variable length (trailing spaces removed)* |
| **BINARY,VARBINARY***[(length)]* | *0…255 – stores bytes instead of character strings* |
| **TINYTEXT\|TINYBLOB** | *0…255 – text stores strings, blob stores bytes* |
| **TEXT\|BLOB** | *0…65,535 – text stores strings, blob stores bytes* |
| **MEDIUMTEXT\|MEDIUMBLOB** | *0…16,777,215 – text stores strings, blob stores bytes* |
| **LONGTEXT\|LONGBLOB** | *0…4,294,967,295 – text stores strings, blob stores bytes* |
| **ENUM***('value1', 'value2',…)* | *list of up to 65,535 members, can have only one value* |
| **SET***('value1', 'value2',…)* | *list of up to 64 members, can have zero or more values* |

**REGEXP** *'expression'*

### functions

| | | |
|---|---|---|
| **ASCII(***'str'***)** | **CONV(***number,from_base,to_base***)** | **BIN(***num***),OCT(***num***),HEX(***num***)** |
| **ORD(***'str'***)** | **CHAR(***number[ USING charset],…***)** | **CONCAT(***'str1', 'str1',…***)** |
| **LENGTH(***'str'***)** | **CHAR_LENGTH(***'str'***)** | **CONCAT_WS(***'separator', 'str1', 'str2'***)** |
| **BIT_LENGTH(***'str'***)** | **REVERSE(***'str'***)** | **SOUNDEX(***'str'***)** |
| **LCASE(***'str'***)** | **UCASE(***'str'***)** | **QUOTE(***'str'***)** |
| **LPAD(***'str', len, 'padstr'***)** | **RPAD(***'str', len, 'padstr'***)** | **ELT(***number, 'str1', 'str2', 'str3',…***)** |
| **LEFT(***'str', length***)** | **RIGHT(***'str', length***)** | **FIELD(***'str', 'str1', 'str2', 'str3',…***)** |
| **LTRIM(***'str'***)** | **RTRIM(***'str'***)** **TRIM(***'str'***)** | **LOAD_FILE(***'filename'***)** |
| **SPACE(***count***)** | **REPEAT(***'str', count***)** | **SUBSTRING(***'str', pos[, length]***)** |
| **REPLACE(***'str', 'from', 'to'***)** | **INSERT(***'str', pos, length, 'newstr'***)** | **SUBSTRING_INDEX(***'str', 'del', count***)** |
| **INSTR(***'str', 'substr'***)** | **LOCATE(***'substr', 'str'[, pos]***)** | **STRCMP(***'str1', 'str2'***)** |

## date & time

| | |
|---|---|
| **DATE** | *'YYYY-MM-DD'* |
| **DATETIME** | *'YYYY-MM-DD HH:MM:SS'* |
| **TIMESTAMP***[(display width)]* | *'YYYY-MM-DD HH:MM:SS' – display widths: 6, 8, 12 or 14* |
| **TIME** | *'HH:MM:SS'* |
| **YEAR***[(2\|4)]* | *'YYYY' – a year in 2-digit or 4-digit format* |

### functions

| | | |
|---|---|---|
| **WEEK(***'date'[, mode]***)** | **WEEKDAY(***'date'***)** | **DAYOFWEEK(***'date'***)** |
| **DAYOFYEAR(***'date'***)** | **MONTH(***'date'***)** | **MONTHNAME(***'date'***)** |
| **QUARTER(***'date'***)** | **YEAR(***'date'***)** | **YEARWEEK(***'date'[, mode]***)** |
| **HOUR(***'date'***)** | **MINUTE(***'date'***)** | **SECOND(***'date'***)** |
| **TO_DAYS(***'date'***)** | **FROM_DAYS(***number***)** | **LAST_DAY(***'date'***)** |
| **SEC_TO_TIME(***seconds***)** | **TIME_TO_SEC(***'time'***)** | **SYSDATE()** |

**CURTIME(),CURRENT_TIME(),CURRENT_TIME** **TIME_FORMAT(***'date', 'format'***)**
**CURDATE(),CURRENT_DATE(),CURRENT_DATE** **DATE_FORMAT(***'date', 'format'***)**

**NOW(),CURRENT_TIMESTAMP(),CURRENT_TIMESTAMP,LOCALTIME(),LOCALTIME**

**UNIX_TIMESTAMP(***['date']***)** **FROM_UNIXTIME(***'unix_timestamp'[, 'format']***)**
**PERIOD_ADD(***'period', num***)** **PERIOD_DIFF(***'period', num***)** **EXTRACT(***unit FROM 'date'***)**

**ADDDATE(***'date', days***) | ADDDATE(***'date', INTERVAL expr unit***),DATE_ADD(***'date', INTERVAL expr unit***)**
**SUBDATE(***'date', days***) | SUBDATE(***'date', INTERVAL expr unit***),DATE_SUB(***'date', INTERVAL expr unit***)**

## commands

**connecting to a database**

*# mysql [-h hostname] [-u username] [-ppassword] [dbname]*

**importing data**

*# mysql dbname < dbdumpfile.sql*

**backup a database**

*# mysqldump [-options] dbname [> dumpfile.sql]*

## syntax & examples

**Create a database**
```
mysql> CREATE DATABASE dbname;
```

**Select a database**
```
mysql> USE dbname;
```

**Delete a database**
```
mysql> DROP DATABASE dbname;
```

**Add a user to a database**
```
mysql> GRANT ALL [PRIVILEGES] ON database.* TO [username]@'hostname' [IDENTIFIED BY 'password'];
```

**List tables in a database**
```
mysql> SHOW TABLES;
```

**Show table format**
```
mysql> DESCRIBE table;
```

**Delete records in a table**
```
mysql> DELETE FROM TABLE table [WHERE conditions];
```

**Create a table**
```
mysql> CREATE TABLE table (column definition,...) [options...];
```

**Show create table syntax**
```
mysql> SHOW CREATE TABLE table;
```

**Change a column definition in a table**
```
mysql> ALTER TABLE table CHANGE column definition;
```

**Add a column to a table**
```
mysql> ALTER TABLE table ADD column definition [AFTER col];
```

**Change auto_increment value**
```
mysql> ALTER TABLE table AUTO_INCREMENT=value;
```

**Alter table syntax**
```
mysql> ALTER TABLE table change specs[, change specs...];
```

**Add a new record**
```
mysql> INSERT table (column1, column2,...) VALUES (expr1, expr2...);
```

**or Add a new record**
```
mysql> INSERT table SET column=expr[, column=expr...);
```

**Update a record in a single table**
```
mysql> UPDATE table SET column=expr[, column=expr...] [WHERE conditions] [ORDER BY ...] [LIMIT count]
```

**Retrieve information from a table**
```
mysql> SELECT {*|expr|column,...} [FROM table,...] [WHERE conditions] [GROUP BY ...] [HAVING conditions] [ORDER BY ...] [LIMIT count]
```

## miscellaneous functions

| | | |
|---|---|---|
| **DATABASE()** | **VERSION()** | **CONNECTION_ID()** |
| **USER()** | **CURRENT_USER()** | **PASSWORD(**'string'**)** |
| **FOUND_ROWS()** | **ROW_COUNT()** | **LAST_INSERT_ID(**[expr]**)** |
| **BIT_COUNT(**number**)** | **FORMAT(**number,digits**)** | **BENCHMARK(**count, expr**)** |
| **CAST(**expr AS type**)** | **CONVERT(**expr, type**)** | **CHARSET(**'str'**)** |
| **INET_NTOA(**expr**)** | **INET_ATON(**expr**)** | **LEAST(**val1,val2,...**)** |
| **GET_LOCK(**'lock',timeout**)** | **RELEASE_LOCK(**'lock'**)** | **GREATEST(**val1,val2,...**)** |
| **ENCRYPT(**'str'[, 'salt']**)** | **DECODE(**'crypt', 'pass'**)** | **ENCODE(**'str', 'password'**)** |
| **MD5(**'string'**)** | **SHA1(**'string'**)** | **AES_ENCRYPT(**'str', 'key'**)** |
| **COMPRESS(**'string'**)** | **UNCOMPRESS(**'string'**)** | **AES_DECRYPT(**'str', 'key'**)** |
| **DES_ENCRYPT(**'str'[, {keynum\|keystr}]**)** | | **DES_DECRYPT(**'string'[, 'key']**)** |

## grouping functions

| | |
|---|---|
| **AVG(**expr**)** | **SUM(**expr**)** |
| **MIN(**expr**)** | **MAX(**expr**)** |
| **VARIANCE(**expr**)** | **STD(**expr**)** |
| **BIT_AND(**expr**)** | **BIT_OR(**expr**)** |

**COUNT(**expr**)**
**COUNT(**DISTINCT expr[, expr...]**)**
**GROUP_CONCAT(**expr**)**

```
GROUP_CONCAT([DISTINCT] expr[, expr...]
        [ORDER BY {int|column|expr}
                [ASC | DESC] [, column ...]
        [SEPARATOR 'string'])
```

## operators

| | |
|---|---|
| **AND, &&** | *Logical AND* |
| **\|\|, OR** | *Logical OR* |
| **XOR** | *Logical XOR* |
| **BINARY** | *Cast a string to binary string* |
| **&** | *Bitwise AND* |
| **\|** | *Bitwise OR* |
| **^** | *Bitwise XOR* |
| **<<** | *Left shift* |
| **>>** | *Right shift* |
| **-** | *Invert bits* |
| **-** | *Change sign of value* |
| **-** | *Minus* |
| **+** | *Addition* |
| ***** | *Multiplication* |
| **%** | *Modulo* |
| **DIV, /** | *Integer division, division* |
| **<=>** | *NULL-safe equal to* |
| **=** | *Equal operator* |
| **>=** | *Greater than or equal to* |
| **>** | *Greater than* |
| **<=** | *Less than or equal to* |
| **<** | *Less than* |
| **IS** | *Boolean test* |
| **LIKE** | *Simple pattern matching* |
| **!=, <>** | *Not equal to* |
| **NOT LIKE** | *Negative simple match* |
| **NOT RGEXP** | *Negative regular expression* |
| **NOT, !** | *Negates value* |
| **REGEXP** | *Match on regular expression* |
| **RLIKE** | *Synonym for REGEXP* |
| **SOUNDS LIKE** | *Compare sounds* |

## control flow

```
IF(expression,true_result,false_result)
IFNULL(expression,result)
NULLIF(expression1,expression2)
CASE [value] WHEN [comparison] THEN [result]
        [WHEN [comparison] THEN result...]
        [ELSE result] END
```

CREATED BY JONATHAN DAVIS
insites.ingenesis.net