Inhaltsverzeichnis

1 Structured Query Language (SQL)

Structured Query Language is a specific language used in programming and designed for managing data held in a relational database management system (RDBMS).

1.1 Connecting to the Server

Use the program putty.exe for connecting from the Windows-Client to the Linux-Server, where the MySQL-Server is running. After logged in you see the Linux-Shell use the command mysql for connecting to the MySQL-Server.

To connect to the server, you will usually need to provide a MySQL user name when you invoke mysql and, most likely, a password. If the server runs on a machine other than the one where you log in, you will also need to specify a host name. Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

Host and user represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The ******** represents your password; enter it when mysql displays the Enter password: prompt.

```
Shell> mysql -h localhost -u user -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g
Your MySQL connection id is 70269
Server version: 5.5.60-0+deb8u1 (Debian)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights
reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.
```

The mysql> prompt tells you that mysql is ready for you to enter SQL statements. If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

After you have connected successfully, you can disconnect any time by typing quit (or \q) at the mysql> prompt:

```
mysql> quit
Bye
```

1.2 Syntax of SQL-Statements

Make sure that you are connected to the server, as discussed in the previous section.

- This following queries illustrates several things about `mysql`:
- `Keywords` may be entered in any lettercase (`uppercase` or `lowercase`).
- A query normally consists of an SQL statement followed by a `semicolon`.
- When you issue a query, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another query.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results.
- `mysql` shows how many rows were returned and how long the query took to execute.

The following table shows each of the prompts you may see and summarizes what they mean:

Prompt	Meaning
`mysql>`	Ready for new query
`->`	Waiting for next line of multiple-line query
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (')
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (")

1.3 MySQL Data Types

In MySQL there are three main data types: `text, number`, and `date`. In addition to the following data types there are `tiny`, `medium` and `long text` data types as well as `tiny`, `small`, `medium` and `big integer` data types. Data types can be entered in `uppercase` or `lowercase`.

1.3.1 Data Types for Text

Important text types are:

`char(size)`	Holds a fixed length string. The fixed size is specified in parenthesis. Can store up to 255 characters.
`varchar(size)`	Holds a variable length string. The maximum size is specified in parenthesis. Can store up to 255 characters.
`text`	Holds a string with a maximum length of 65535 characters.
`blob`	Binary Large Objects holds up to 65535 bytes of data

1.3.2 Data Types for Numbers

Important number types are:

`int(size)`	The maximum number of digits may be specified in parenthesis. The possible values are from -2147483648 to 2147483647.
`float(size,d)`	A small number with a floating decimal point is stored. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter.
`double(size,d)`	A large number with a floating decimal point is stored. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter.
`decimal(size,d)`	A double is stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter

1.3.3 Data Types for the Date

The date types are:

`date`	Stores a date in the format: YYYY-MM-DD The supported range is from '1000-01-01' to '9999-12-31'
`datetime`	This data type is used for values that contain both date and time parts in 'YYYY-MM-DD HH:MI:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59' .
`timestamp`	Timestamp type values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC) in the format 'YYYY-MM-DD HH:MI:SS'. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
`time`	This data type is used for time values in the format 'HH:MI:SS'. The supported range is from '-838:59:59' to '838:59:59'
`year`	This data type stores a year in two-digit or four-digit format. The allowed values in four-digit format are 1901 to 2155. The allowed values in two-digit format: 70 to 69, representing years from 1970 to 2069

1.4 Some SQL-Statements

1.4.1 Version Number and Current Date

Here is a simple query that asks the server to tell you the MySQL-version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> select version(), current_date;
+-----------------+--------------+
| VERSION()       | CURRENT_DATE |
+-----------------+--------------+
| 5.5.60-0+deb8u1 | 2018-09-13   |
+-----------------+--------------+
```

1.4.2 Use MySQL as Calculator

You can even enter multiple statements on a single line. Just end each line with a semicolon:

```
mysql> select sin(pi()/4), (4+1)*5;
+--------------------+---------+
| sin(pi()/4)        | (4+1)*5 |
+--------------------+---------+
| 0.7071067811865475 |      25 |
+--------------------+---------+
```

1.5 Basic SQL-Statements

1.5.1 Show Databases

Use the following statement to find out what databases currently exist on the server for the current user:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| userdb             |
+--------------------+
```

1.5.2 Select Database

Your database needs to be created only once, but you must select it for use each time you begin a mysql session. To make `userdb` the current database, use this statement:

```
mysql> use userdb;
```

Alternatively, you can select the database on the command line when you invoke mysql:

```
shell> mysql -h host -u user -p userdb
Enter password: ********
```

1.5.3 Show Tables

Use the following statement to find out what tables currently exist on the server for the current user:

```
mysql> show tables;
Empty set (0.00 sec)
```

1.5.4 Create Table

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the pet table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays

- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the pet table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.
Use the following statement to specify the layout of your table pet:

```
mysql> create table pet (
    ->    name varchar(20),
    ->    owner varchar(20),
    ->    species varchar(20),
    ->    sex char(1),
    ->    birth date,
    ->    death date);
Query OK, 0 rows affected (0.02 sec)
```

The data type varchar is a good choice for the name, owner, and species columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be 20.

Several types of values can be chosen to represent sex in animal records, such as 'm' and 'f', or perhaps 'male' and 'female'. It is simplest to use the single characters 'm' and 'f'.

The use of the data type date for the birth and death columns is a fairly obvious choice.

1.5.5 Show Table Structure

To verify that your table `pet` was created the way you expected, use the `describe` statement:

```
mysql> describe pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

1.5.6 Insert Table Record

When you want to add new records one at a time, the `insert` statement is useful.

```
mysql> insert into pet
 ->  values ('Puffball','Diane','hamster','f','2018-03-30','2018-04-11');
Query OK, 1 row affected (0.00 sec)
```

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table.

1.5.7 Show Table Content

The simplest form of the `select` statement retrieves everything from the table `pet`:

```
mysql> select * from pet;
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Puffball | Diane | hamster | f    | 2018-03-30 | 2018-04-11 |
+----------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)
```

1.5.8 Delete Whole Table Content

The simplest form of the `delete` statement removes the entire content from the table `pet` is:

```
mysql> delete from pet;
Query OK, 1 row affected (0.01 sec)
```

1.5.9 Remove Table

The fast way of removing the table pet is:

```
mysql> drop table pet;
Query OK, 0 rows affected (0.03 sec)
```

1.6 Modify the Table Structure

First create the table `pet` and at least one record as shown before.

1.6.1 Change Table Name

The statement for changing the table name from `pet` to `animal` is:

```
mysql> alter table pet
    -> rename animal;
Query OK, 0 rows affected (0.00 sec)
```

1.6.2 Add Table Column

To add the column `size` with data type `int` in the table `animal`, use the following syntax:

```
mysql> alter table animal
    -> add size int;
Query OK, 1 row affected (0.03 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

Now the structure of the table `animal` has changed.

```
mysql> describe animal;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
| size    | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

1.6.3 Modify Name of Table Column

To change the name of the column from `size` to `weight` use the following syntax:

```
mysql> alter table animal
    -> change size weight int;
Query OK, 1 row affected (0.05 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

1.6.4 Modify Data Type of Table Column

To change the data type of the column from `int` to `double` use the statement:

```
mysql> alter table animal
    -> change weight weight double;
Query OK, 1 row affected (0.08 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

1.6.5 Drop Table Column

The following statement deletes the column `owner` from the table `animal`:

```
mysql> alter table animal
    -> drop owner;
Query OK, 1 row affected (0.10 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

Now the structure of the table animal has changed.

```
mysql> describe animal;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
| weight  | double      | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

The content of the table `animal` is following:

```
mysql> select * from animal;
+----------+---------+------+------------+------------+--------+
| name     | species | sex  | birth      | death      | weight |
+----------+---------+------+------------+------------+--------+
| Puffball | hamster | f    | 2018-03-30 | 2018-04-11 |   NULL |
+----------+---------+------+------------+------------+--------+
1 row in set (0.00 sec)
```

1.7 Change the Table Content

First we add an additional record into the table `animal`:

```
mysql> insert into animal
    -> values ("Fluffy","cat","f","2017-02-16",NULL,2.75);
Query OK, 1 row affected (0.50 sec)
```

1.7.1 Update Table Column

To update the `weight` and `species` in the record of the `Puffball` use the update statement with the appropriate where clause:

```
mysql> update animal set weight=0.5,species="frog"
    -> where name="Puffball";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

The content of the table `animal` has changed:

```
mysql> select * from animal;
+----------+---------+------+------------+------------+--------+
| name     | species | sex  | birth      | death      | weight |
+----------+---------+------+------------+------------+--------+
| Puffball | frog    | f    | 2018-03-30 | 2018-04-11 |    0.5 |
| Fluffy   | cat     | f    | 2017-02-16 | NULL       |   2.75 |
+----------+---------+------+------------+------------+--------+
```

1.7.2 Delete Table Record

To delete the entire record of the cat `Fluffy` use the delete statement with the appropriate where clause:

```
mysql> delete from animal
    -> where name="Fluffy";
Query OK, 1 row affected (0.07 sec)
```

1.8 Primary Key

A `primary key` in the table is used to ensure data in the specific column is unique. Thus each record in the table becomes unique. Finally all relationships between tables are established by using the primary keys of all concerned tables.

The column used as primary key `must not be empty` und `has to be unique`. Commonly the data type is some kind of `int`. Additionally `auto increment` allows a unique number to be `generated automatically` when a new record is inserted into the table.

1.8.1 Create Table

Unless the table animal does not exist, the new table animal can be created with primary key:

```
create table animal(
    id int not null auto_increment,
    name varchar(20),
    species varchar(20),
    sex char(1),
    birth date,
    death date,
    weight double,
    primary key(id));
```

1.8.2 Insert Table Record

Some records can be inserted using the following but different statements:

```
insert into animal
   values (NULL,'Fluffy','cat','f','2003-02-04',NULL, 3.5);
```

If not all columns of the record are filled in, the filled in columns must be specified before:

```
insert into animal (name,species,sex,birth,weight)
   values ('Claws','cat','m','2004-03-17',4.9);
```

Another records are inserted too:

```
insert into animal
   values (NULL,'Buffy','dog','f','2009-05-13',NULL, 22.8);

insert into animal
   values (NULL,'Fang','dog','m','2010-08-27',NULL, 45.1);

insert into animal
   values (NULL,'Bowser','dog','m','1999-08-31','2013-07-29', 33.4);

insert into animal
   values (NULL,'Chirpy','bird','f','2008-09-11',NULL, 0.05);

insert into animal
   values (NULL,'Whistler','bird',NULL,'2007-12-09',NULL, 0.8);

insert into animal
   values (NULL,'Slim','snake','m','2006-04-29',NULL, 27.8);

insert into animal
   values (NULL,'Puffball','hamster','f','2009-03-30',NULL, 0.03);
```

The `structure` of the table `animal` can be displayed:

```
mysql> describe animal;

+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| id      | int(11)     | NO   | PRI | NULL    | auto_increment |
| name    | varchar(20) | YES  |     | NULL    |                |
| species | varchar(20) | YES  |     | NULL    |                |
| sex     | char(1)     | YES  |     | NULL    |                |
| birth   | date        | YES  |     | NULL    |                |
| death   | date        | YES  |     | NULL    |                |
| weight  | double      | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

Now the field (column) `id` is displayed as the `primary key` of the table `animal`.

1.9 Retrieving Information from a Table

The `select` statement is used to pull information from a table.

1.9.1 Selecting All Data

The simplest form of `select` retrieves `everything` from a table:

```
mysql> select * from animal;
+----+----------+---------+------+------------+------------+--------+
| id | name     | species | sex  | birth      | death      | weight |
+----+----------+---------+------+------------+------------+--------+
|  1 | Fluffy   | cat     | f    | 2003-02-04 | NULL       |    3.5 |
|  2 | Claws    | cat     | m    | 2004-03-17 | NULL       |    4.9 |
|  3 | Buffy    | dog     | f    | 2009-05-13 | NULL       |   22.8 |
|  4 | Fang     | dog     | m    | 2010-08-27 | NULL       |   45.1 |
|  5 | Bowser   | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
|  6 | Chirpy   | bird    | f    | 2008-09-11 | NULL       |   0.05 |
|  7 | Whistler | bird    | NULL | 2007-12-09 | NULL       |    0.8 |
|  8 | Slim     | snake   | m    | 2006-04-29 | NULL       |   27.8 |
|  9 | Puffball | hamster | f    | 2009-03-30 | NULL       |   0.03 |
+----+----------+---------+------+------------+------------+--------+
```

1.9.2 Selecting Particular Rows

You can select only particular rows from your table. Use the `where` clause for the desired record(s).

To display all columns of the cat `Fluffy`:

```
mysql> select * from animal where name = 'Fluffy';
+----+--------+---------+------+------------+-------+--------+
| id | name   | species | sex  | birth      | death | weight |
+----+--------+---------+------+------------+-------+--------+
|  1 | Fluffy | cat     | f    | 2003-02-04 | NULL  |    3.5 |
+----+--------+---------+------+------------+-------+--------+
```

String comparisons normally are `case-insensitive`, so you can specify the name as 'Fluffy', 'FLUFFY', and so forth. The query result is the same.

You can specify conditions on any column, not just name. For example, if you want to know which animals were born during or after 2009, test the birth column:

```
mysql> select * from animal where birth >= '2009-1-1';
+----+----------+---------+------+------------+-------+--------+
| id | name     | species | sex  | birth      | death | weight |
+----+----------+---------+------+------------+-------+--------+
|  3 | Buffy    | dog     | f    | 2009-05-13 | NULL  |   22.8 |
|  4 | Fang     | dog     | m    | 2010-08-27 | NULL  |   45.1 |
|  9 | Puffball | hamster | f    | 2009-03-30 | NULL  |   0.03 |
+----+----------+---------+------+------------+-------+--------+
```

You can combine conditions, for example using the `AND` operator, to locate female birds:

```
mysql> select * from animal
    -> where species = 'bird' and sex = 'f';
+----+--------+---------+------+------------+-------+--------+
| id | name   | species | sex  | birth      | death | weight |
+----+--------+---------+------+------------+-------+--------+
|  6 | Chirpy | bird    | f    | 2008-09-11 | NULL  |   0.05 |
+----+--------+---------+------+------------+-------+--------+
```

There is also an `OR` operator:

```
mysql> select * from animal
    -> where species = 'cat' or species = 'dog';
+----+--------+---------+------+------------+------------+--------+
| id | name   | species | sex  | birth      | death      | weight |
+----+--------+---------+------+------------+------------+--------+
|  1 | Fluffy | cat     | f    | 2003-02-04 | NULL       |    3.5 |
|  2 | Claws  | cat     | m    | 2004-03-17 | NULL       |    4.9 |
|  3 | Buffy  | dog     | f    | 2009-05-13 | NULL       |   22.8 |
|  4 | Fang   | dog     | m    | 2010-08-27 | NULL       |   45.1 |
|  5 | Bowser | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
+----+--------+---------+------+------------+------------+--------+
```

AND and OR may be intermixed, although AND has higher precedence than OR. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> select * from animal
    -> where (species = 'dog' and sex = 'm')
    -> or (species = 'hamster' and sex = 'f');
+----+----------+---------+------+------------+------------+--------+
| id | name     | species | sex  | birth      | death      | weight |
+----+----------+---------+------+------------+------------+--------+
|  4 | Fang     | dog     | m    | 2010-08-27 | NULL       |   45.1 |
|  5 | Bowser   | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
|  9 | Puffball | hamster | f    | 2009-03-30 | NULL       |   0.03 |
+----+----------+---------+------+------------+------------+--------+
```

To find the living animals:

```
mysql> select * from animal where death is NULL;
+----+----------+---------+------+------------+-------+--------+
| id | name     | species | sex  | birth      | death | weight |
+----+----------+---------+------+------------+-------+--------+
|  1 | Fluffy   | cat     | f    | 2003-02-04 | NULL  |    3.5 |
|  2 | Claws    | cat     | m    | 2004-03-17 | NULL  |    4.9 |
|  3 | Buffy    | dog     | f    | 2009-05-13 | NULL  |   22.8 |
|  4 | Fang     | dog     | m    | 2010-08-27 | NULL  |   45.1 |
|  6 | Chirpy   | bird    | f    | 2008-09-11 | NULL  |   0.05 |
|  7 | Whistler | bird    | NULL | 2007-12-09 | NULL  |    0.8 |
|  8 | Slim     | snake   | m    | 2006-04-29 | NULL  |   27.8 |
|  9 | Puffball | hamster | f    | 2009-03-30 | NULL  |   0.03 |
+----+----------+---------+------+------------+-------+--------+
```

To find the dead animals:

```
mysql> select * from animal where death is not null;
+----+--------+---------+------+------------+------------+--------+
| id | name   | species | sex  | birth      | death      | weight |
+----+--------+---------+------+------------+------------+--------+
|  5 | Bowser | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
+----+--------+---------+------+------------+------------+--------+
```

So you can see, the date value NULL is not case-sensitive.

1.9.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas.

For example, if you want to know when your animals were born, select the name and birth columns:

```
mysql> select name, birth from animal;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 2003-02-04 |
| Claws    | 2004-03-17 |
| Buffy    | 2009-05-13 |
| Fang     | 2010-08-27 |
| Bowser   | 1999-08-31 |
| Chirpy   | 2008-09-11 |
| Whistler | 2007-12-09 |
| Slim     | 2006-04-29 |
| Puffball | 2009-03-30 |
+----------+------------+
```

To find out the sex of the animals, use this query:

```
mysql> select sex from animal;
+------+
| sex  |
+------+
| f    |
| m    |
| f    |
| m    |
| m    |
| f    |
| NULL |
| m    |
| f    |
+------+
```

Notice that the query simply retrieves the sex column from each record, thus the sex appears more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `distinct`:

```
mysql> select distinct sex from animal;
+------+
| sex  |
+------+
| f    |
| m    |
| NULL |
+------+
```

You can use a `where` clause to combine row selection with column selection.

For example, to get birth dates for dogs and cats only, use this query:

```
mysql> select name, species, birth from animal
    -> where species = 'dog' or species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 2003-02-04 |
| Claws  | cat     | 2004-03-17 |
| Buffy  | dog     | 2009-05-13 |
| Fang   | dog     | 2010-08-27 |
| Bowser | dog     | 1999-08-31 |
+--------+---------+------------+
```

1.9.4 Sorting Rows

To sort a result, use an `order by` clause. Here are animal birthdays, sorted by birth date:

```
mysql> select name, birth from animal order by birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Bowser   | 1999-08-31 |
| Fluffy   | 2003-02-04 |
| Claws    | 2004-03-17 |
| Slim     | 2006-04-29 |
| Whistler | 2007-12-09 |
| Chirpy   | 2008-09-11 |
| Puffball | 2009-03-30 |
| Buffy    | 2009-05-13 |
| Fang     | 2010-08-27 |
+----------+------------+
```

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `desc` keyword to the name of the column you are sorting by:

```
mysql> select name, birth from animal order by birth desc;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fang     | 2010-08-27 |
| Buffy    | 2009-05-13 |
| Puffball | 2009-03-30 |
| Chirpy   | 2008-09-11 |
| Whistler | 2007-12-09 |
| Slim     | 2006-04-29 |
| Claws    | 2004-03-17 |
| Fluffy   | 2003-02-04 |
| Bowser   | 1999-08-31 |
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions.

For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> select name, species, birth from animal
    -> order by species asc, birth desc;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 2008-09-11 |
| Whistler | bird    | 2007-12-09 |
| Claws    | cat     | 2004-03-17 |
| Fluffy   | cat     | 2003-02-04 |
| Fang     | dog     | 2010-08-27 |
| Buffy    | dog     | 2009-05-13 |
| Bowser   | dog     | 1999-08-31 |
| Puffball | hamster | 2009-03-30 |
| Slim     | snake   | 2006-04-29 |
+----------+---------+------------+
```

1.10 Working with NULL-Values

`NULL` means “a missing unknown value” and it is treated differently from other values. To test for `NULL` (or `null`), use the `is null` and `is not null` operators.

To find all living dogs use the statement:

```
mysql> select * from animal
    -> where species='dog' and death is null;
+----+-------+---------+------+------------+-------+--------+
| id | name  | species | sex  | birth      | death | weight |
+----+-------+---------+------+------------+-------+--------+
|  3 | Buffy | dog     | f    | 2009-05-13 | NULL  |   22.8 |
|  4 | Fang  | dog     | m    | 2010-08-27 | NULL  |   45.1 |
+----+-------+---------+------+------------+-------+--------+
```

To find all dead dogs use the statement:

```
mysql> select * from animal
    -> where species='dog' and death is not null;
+----+--------+---------+------+------------+------------+--------+
| id | name   | species | sex  | birth      | death      | weight |
+----+--------+---------+------+------------+------------+--------+
|  5 | Bowser | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
+----+--------+---------+------+------------+------------+--------+
```

1.11 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates.

For example, to calculate ages or extract parts of dates.

1.11.1 Calculate the Age in Years, Months or Days from a Date

To determine how many years old each of your cats is, use the `timestampdiff()` function.

```
mysql>select name, birth, curdate(),
    ->   timestampdiff(year,birth,curdate()) as AgeInYears
    ->   from animal where species='cat';
+--------+------------+------------+------------+
| name   | birth      | curdate()  | AgeInYears |
+--------+------------+------------+------------+
| Fluffy | 2003-02-04 | 2018-09-30 |         15 |
| Claws  | 2004-03-17 | 2018-09-30 |         14 |
+--------+------------+------------+------------+
```

To determine how many month old each of your dogs is. You can sort the younger first:

```
mysql> select name, birth, curdate(),
    -> timestampdiff(month,birth,curdate()) as AgeInMonths
    -> from animal where species='dog' order by AgeInMonths;
+--------+------------+------------+-------------+
| name   | birth      | curdate()  | AgeInMonths |
+--------+------------+------------+-------------+
| Fang   | 2010-08-27 | 2018-09-30 |          97 |
| Buffy  | 2009-05-13 | 2018-09-30 |         112 |
| Bowser | 1999-08-31 | 2018-09-30 |         228 |
+--------+------------+------------+-------------+
```

To determine how many days old each of your birds is. You can sort the older first:

```
mysql> select name, birth, curdate(),
    -> timestampdiff(day,birth,curdate()) as AgeInDays
    -> from animal where species='bird' order by AgeInDays desc;
+----------+------------+------------+-----------+
| name     | birth      | curdate()  | AgeInDays |
+----------+------------+------------+-----------+
| Whistler | 2007-12-09 | 2018-09-30 |      3948 |
| Chirpy   | 2008-09-11 | 2018-09-30 |      3671 |
+----------+------------+------------+-----------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the death value is NULL. Then, for those with non-NULL values, compute the difference between the death and birth values:

```
mysql> select name, birth, death,
    -> timestampdiff(year,birth,death) as age
    -> from animal where death is not null order by age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1999-08-31 | 2013-07-29 |   13 |
+--------+------------+------------+------+
```

1.11.2 Extract the Year, Month or Day from a Date

MySQL provides several functions for extracting parts of dates: `year()`, `month()` and `dayofmonth()`.

To extract the month of the birth date for the snake use:

```
mysql> select name, birth, month(birth) from animal
    -> where species='snake';
+------+------------+--------------+
| name | birth      | month(birth) |
+------+------------+--------------+
| Slim | 2006-04-29 |            4 |
+------+------------+--------------+
```

To extract the birthday for the snake use:

```
mysql> select name, birth, dayofmonth(birth) as birthday from animal
    -> where species='snake';
+------+------------+----------+
| name | birth      | birthday |
+------+------------+----------+
| Slim | 2006-04-29 |       29 |
+------+------------+----------+
```

Finding animals with birthdays in the upcoming month is simple. Suppose that the current month is July. Then the month value is 7 and you can look for animals born in August:

```
mysql> select name, species, birth from animal where month(birth) = 8;
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fang   | dog     | 2010-08-27 |
| Bowser | dog     | 1999-08-31 |
+--------+---------+------------+
```

Finding animals with birthdays in this month use the following where clause:

```
mysql> select * from animal
    -> where month(curdate())=month(birth);
```

1.12 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on regular expressions similar to those used by Unix utilities.

1.12.1 SQL Pattern Matching

SQL pattern matching enables you to use _ to match any single character and % to match an arbitrary number of characters. Do not use = or <> when you use SQL patterns. Use the like or not like comparison operators instead.

To find names beginning with b:

```
mysql> select * from animal
    -> where name like 'b%';
+----+--------+---------+------+------------+------------+--------+
| id | name   | species | sex  | birth      | death      | weight |
+----+--------+---------+------+------------+------------+--------+
|  3 | Buffy  | dog     | f    | 2009-05-13 | NULL       |   22.8 |
|  5 | Bowser | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
+----+--------+---------+------+------------+------------+--------+
```

To find names ending with fy:

```
mysql> select * from animal
    -> where name like '%fy';
+----+--------+---------+------+------------+-------+--------+
| id | name   | species | sex  | birth      | death | weight |
+----+--------+---------+------+------------+-------+--------+
|  1 | Fluffy | cat     | f    | 2003-02-04 | NULL  |    3.5 |
|  3 | Buffy  | dog     | f    | 2009-05-13 | NULL  |   22.8 |
+----+--------+---------+------+------------+-------+--------+
```

To find names containing a w:

```
mysql> select * from animal
    -> where name like '%w%';
+----+----------+---------+------+------------+------------+--------+
| id | name     | species | sex  | birth      | death      | weight |
+----+----------+---------+------+------------+------------+--------+
|  2 | Claws    | cat     | m    | 2004-03-17 | NULL       |    4.9 |
|  5 | Bowser   | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
|  7 | Whistler | bird    | NULL | 2007-12-09 | NULL       |    0.8 |
+----+----------+---------+------+------------+------------+--------+
```

To find names containing exactly five characters, use five instances of the _ pattern character:

```
mysql> select * from animal
    -> where name like '_____';
+----+-------+---------+------+------------+-------+--------+
| id | name  | species | sex  | birth      | death | weight |
+----+-------+---------+------+------------+-------+--------+
|  2 | Claws | cat     | m    | 2004-03-17 | NULL  |    4.9 |
|  3 | Buffy | dog     | f    | 2009-05-13 | NULL  |   22.8 |
+----+-------+---------+------+------------+-------+--------+
```

To find names containing an u as the second letter:

```
mysql> select * from animal
    -> where name like '_u%';
+----+----------+---------+------+------------+-------+--------+
| id | name     | species | sex  | birth      | death | weight |
+----+----------+---------+------+------------+-------+--------+
|  3 | Buffy    | dog     | f    | 2009-05-13 | NULL  |   22.8 |
|  9 | Puffball | hamster | f    | 2009-03-30 | NULL  |   0.03 |
+----+----------+---------+------+------------+-------+--------+
```

1.12.2 Regular Expressions Pattern Matching

The other type of pattern matching provided by MySQL uses regular expressions. When you test for a match for this type of pattern, use the regexp and not regexp operators (or rlike and not rlike, which are synonyms).

The following list describes some characteristics of regular expressions:

. matches any single character.

A character class [...] matches any character within the brackets. For example, [abc] matches a, b, or c. To name a range of characters, use a dash. [a-z] matches any letter, whereas [0-9] matches any digit.

* matches zero or more instances of the thing preceding it. For example, x* matches **any** number of x characters, [0-9]* matches any number of digits, and .* matches any number of anything.

A regular expression pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a like pattern match, which succeeds only if the pattern matches the entire value.)

To anchor a pattern so that it must match the beginning or end of the value being tested, use ^ at the beginning or $ at the end of the pattern.

To demonstrate how regular expressions work, the like queries shown previously are rewritten here to use regexp (= rlike).

To find names beginning with b, use ^ to match the beginning of the name:

```
mysql> select * from animal where name regexp '^b';
+----+--------+---------+------+------------+------------+--------+
| id | name   | species | sex  | birth      | death      | weight |
+----+--------+---------+------+------------+------------+--------+
|  3 | Buffy  | dog     | f    | 2009-05-13 | NULL       |   22.8 |
|  5 | Bowser | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
+----+--------+---------+------+------------+------------+--------+
```

The same result using:

```
mysql> select * from animal where name rlike '^b';
```

To force a regexp comparison to be case-sensitive, use the binary keyword to make one of the strings a binary string.

This query matches only lowercase b at the beginning of a name:

```
mysql> select * from animal where name regexp binary '^b';
Empty set (0.00 sec)
```

To find names ending with fy, use $ to match the end of the name:

```
mysql> select * from animal where name regexp 'fy$';
+----+--------+---------+------+------------+-------+--------+
| id | name   | species | sex  | birth      | death | weight |
+----+--------+---------+------+------------+-------+--------+
|  1 | Fluffy | cat     | f    | 2003-02-04 | NULL  |    3.5 |
|  3 | Buffy  | dog     | f    | 2009-05-13 | NULL  |   22.8 |
+----+--------+---------+------+------------+-------+--------+
```

To find names containing a w, use this query:

```
mysql> select * from animal where name regexp 'w';
+----+----------+---------+------+------------+------------+--------+
| id | name     | species | sex  | birth      | death      | weight |
+----+----------+---------+------+------------+------------+--------+
|  2 | Claws    | cat     | m    | 2004-03-17 | NULL       |    4.9 |
|  5 | Bowser   | dog     | m    | 1999-08-31 | 2013-07-29 |   33.4 |
|  7 | Whistler | bird    | NULL | 2007-12-09 | NULL       |    0.8 |
+----+----------+---------+------+------------+------------+--------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the next query to put a wildcard on either side of the pattern.

To find names containing exactly five characters, use `^` and `$` to match the beginning and end of the name, and five instances of `.` in between:

```
mysql> select * from animal where name rlike '^.....$';
+----+-------+---------+------+------------+-------+--------+
| id | name  | species | sex  | birth      | death | weight |
+----+-------+---------+------+------------+-------+--------+
|  2 | Claws | cat     | m    | 2004-03-17 | NULL  |    4.9 |
|  3 | Buffy | dog     | f    | 2009-05-13 | NULL  |   22.8 |
+----+-------+---------+------+------------+-------+--------+
```

You could also write the previous query using the `{n}` ("repeat-`n`-times ") - operator:

```
mysql> select * from animal where name rlike '^.{5}$';
+----+-------+---------+------+------------+-------+--------+
| id | name  | species | sex  | birth      | death | weight |
+----+-------+---------+------+------------+-------+--------+
|  2 | Claws | cat     | m    | 2004-03-17 | NULL  |    4.9 |
|  3 | Buffy | dog     | f    | 2009-05-13 | NULL  |   22.8 |
+----+-------+---------+------+------------+-------+--------+
```

If additionally the first letter must be a `b`:

```
mysql> select * from animal where name rlike '^b.{4}$';
+----+-------+---------+------+------------+-------+--------+
| id | name  | species | sex  | birth      | death | weight |
+----+-------+---------+------+------------+-------+--------+
|  3 | Buffy | dog     | f    | 2009-05-13 | NULL  |   22.8 |
+----+-------+---------+------+------------+-------+--------+
```

1.13 Counting Rows

Databases are often used to answer the question: “How often does a certain type of data occur in a table?” For example, counting the total number of animals you have is the same question as “How many rows are in the animal table?” because there is one record per animal. `count(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> select count(*) from animal;
+----------+
| count(*) |
+----------+
|        9 |
+----------+
```

The use of `count()` in conjunction with `group by` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Show the number of animals per species:

```
mysql> select species, count(*) from animal group by species;
+---------+----------+
| species | count(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Show the number of animals per sex:

```
mysql> select sex, count(*) from animal group by sex;
+------+----------+
| sex  | count(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

Show the number of animals per combination of species and sex:

```
mysql> select species, sex, count(*) from animal group by species, sex;
+---------+------+----------+
| species | sex  | count(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use count(). For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> select species, sex, count(*) from animal
    -> where species = 'dog' or species = 'cat'
    -> group by species, sex;
+---------+------+----------+
| species | sex  | count(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```