MySQL/Print version

MySQL

Contents

Introduction

What is SQL?

For a more general introduction see the SQL Wikibook.

Structured **Q**uery **L**anguage is a third generation language for working with relational databases. Being a 3G language it is closer to human language than machine language and therefore easier to understand and work with.

- Dr. E. F. Ted Codd who worked for IBM described a relational model for database in 1970.
- In 1992, ANSI (American National Standards Institute), the apex body, standardized most of the basic syntax.
- Its called SQL 92 and most databases (like Oracle, MySQL, Sybase, etc.) implement a subset of the standard (and proprietary extensions that makes them often incompatible).

Why MySQL?

- Free as in Freedom - Released with GPL version 2 license (though a different license can be bought from Oracle, see below)
- Cost - Free!
- Support - Online tutorials, forums, mailing list (lists.mysql.com), paid support contracts.
- Speed - One of the fastest databases available. ([2] (http://www.mysql.com/why-mysql/benchmarks/))
- Functionality - supports most of ANSI SQL commands.
- Ease of use - less need of training / retraining.
- Portability - easily import / export from Excel and other databases
- Scalable - Useful for both small as well as large databases containing billions of records and terabytes of data in hundreds of thousands of tables.
- Permission Control - selectively grant or revoke permissions to users.

The MySQL license

MySQL is available under a *dual-licensing* scheme:

- Under the GNU General Public License, version 2, ("or later" allowed in versions released before 2007): this is a Free (as in freedom), copyleft software license that allows you to use MySQL for commercial and non-commercial purposes in your application, as long as your application is released under the GNU GPL. There is also a "FLOSS Exception" which essentially allows non-GPL'd but Free applications (such as the PHP programming language, under the PHP license) to connect to a MySQL server. The exception lists a set of free and open-source software license that can be used in addition to the GNU GPL for your MySQL-dependent Free application.
- A so-called "commercial" [1], paid license, that is, a license where MySQL grants you the right to integrate MySQL with a non-FLOSS application that you are redistributing outside your own organization. [2]

MySQL and its forks

MySQL is Free Software, so some forks and unofficial builds delivering contributions from the community exist.

MariaDB

In 2008 Sun Microsystems bought MySQL, Sun being itself later acquired by Oracle, in 2010. After the acquisition, the development process has changed. The team has started to release new MySQL versions less frequently, so the new code is less tested.There were also less contributions from the community.

In 2009 Monty Widenius, the founder of MySQL, left the company and created a new one, called The Monty Program (http://www.askmonty.org/). He started a new fork called MariaDB. The scopes of MariaDB,

- import all the new code that will be added to the main MySQL branch, but enhancing it to make it more stable;
- clean the MySQL code;
- add contributions from the community (new plugins, new features);
- develop the Aria storage engine, formerly named Maria;
- improving the performance;
- adding new features to the server.

The license is the GNU GPLv2 (inherited from MySQL).

The primary platform for MariaDB is GNU/Linux, but also works on one proprietary system. The following Storage Engine have been added:

- Aria (also used for internal tables)
- PBXT
- XtraDB
- FederatedX
- SphinxSE
- OQGRAPH
- Others may be added in the future.

Drizzle

In 2008 Brian Aker, chief architect of MySQL, left the project to start a new fork called Drizzle (http://www.drizzle.org/). While Oracle initially funded the project, Drizzle is now funded by Rackspace. Its characteristics are:

- only a small part of the MySQL code has survived in this fork, the rest being removed: only essential features are implemented in the Drizzle server;
- the survived code has been cleaned;
- Drizzle is modular: many features are or can be implemented as plugins;
- the software is optimized for multiCPU and multicore 64 bit machines;
- only GNU/Linux and UNIX systems are supported.

There are no public releases of this fork, still. Its main license will be the GNU GPLv2 (inherited from MySQL), but where possible the BSD license is applied.

OurDelta

OurDelta (http://ourdelta.org/) is another fork, maintained by Open Query. The first branch, which has number 5.0, is based on MySQL 5.0. The 5.1 branch is based on MariaDB. OurDelta includes some patches developed by the community or by third parties. OurDelta provides packages for some GNU/Linux distributions: Debian, Ubuntu, Red Hat/CentOS. It is not available for other systems, but the source code is freely available.

Percona Server

Percona Server is a MySQL fork maintained by Percona. It provides the ExtraDB Storage Engine, which is a fork of InnoDB, and some patches which mainly improve the performance.

Notes

1. Calling it "commercial" is misleading, because the GNU GPL can be used in commercial (but non-proprietary) projects.
2. Proprietary projects still can connect to a MySQL server without purchasing this license by using old versions of the MySQL client connection libraries (under the GNU Lesser General Public License). However, these libraries cannot connect to the newest versions of the MySQL server.

MySQL Practical Guide

Installing MySQL

All in one solutions

As MySQL alone isn't enough to run a real database server, the more practical way to install it is to deploy an all in one pack in this purpose, including all the needed additional elements: Apache and PHP.

1. On Linux: XAMP or LAMP.
2. On Windows: XAMP, WAMP, or EasyPHP.

Attention on Windows 10:

- The server IIS is launched by default, which forces Apache to change its port (888 instead of 80). To resolve this, just untick *Internet Information Services* in *Programs and functionalities, Activate or deactivate the Windows functionalities.* In the same way, the MySQL port can change from 3306 to 3388.
- Moreover, *EasyPHP development server* (alias *Devserver*, the red version) doesn't work properly (*MSVCR110.dll is missing*) but *EasyPHP hosting server* (alias *Webserver*, the blue one) yes. However, it launched automatically at each boot which slows the system significantly. To avoid this, execute *services.msc*, and toggle the three services below in manual start. Then to launch them on demand (as an administrator), create a script called *MySQL.cmd*, containing the following lines:

```
net start ews-dbserver
net start ews-httpserver
net start ews-dashboard
pause
net stop ews-dashboard
net stop ews-httpserver
net stop ews-dbserver
```

Single installation

This guide is written from the perspective of using the Linux Shell with Ubuntu and apt-get[3] (http://www.zolved.com/synapse/view_content/27986/How_to_install_MySQL_On_Ubuntu).

If you want to solely use the Terminal:

Make sure you have the MySQL Client and Server installed.

(Just to be safe.)

```
apt-get install mysql-client mysql-client-5.0 mysql-server mysql-server-5.0
```

About the MySQL package:

[4] (http://www.webdevelopersnotes.com/tutorials/sql/mysql_database_introduction_mysql_beginners_tutorial.php3)

Having a secure installation:

If all your answers are "yes" to what follows, this cleans up your installation, forces you to set a root password, asks you to test for anonymous users and makes your database internal.

Just be careful. Be sure that you are configuring MySQL to the specifications you want.

Here's the code:

```
mysql_secure_installation
```

Creating your own MySQL account and database:

Now that MySQL is installed, you wouldn't necessarily have your own account, so you have to log in as root.

To do this type:

```
sudo mysql -u root -p
```

(This means that you're logging on as the user "root" (**-u root**) and that you're requesting the password for "root" (**-p**))

Once you've managed to log in, your command-line should look like this: **mysql>**

By the way, if your command-line ends up looking like this: -> theres an explanation behind it.

In MySQL each command you do has to end with **;** . This way it knows that everything behind **;** is a command.

So to get out of there, simply type **;** There will be more on this later.

Now you can check what databases (if any) are available to your user (in this case "root"):

```
show databases;
```

Let's get straight to the chase and create our own database. Let's call it **people**. While we're doing this we can also create our own user account. Two birds with one stone.

So first create the database:

```
create database people;
```

(NOTE: in this particular case, you have to be "root" to create new databases.)

Now we want to grant (**GRANT**) all user rights (**ALL**) from (**ON**) the entire (*****) **people** database to (**TO**) your account (*yourusername***@localhost**) with your user password being *stuffedpoodle* (**IDENTIFIED BY "stuffedpoodle"**).

So we'd input this as:

```
GRANT ALL ON people.* TO yourusername@localhost IDENTIFIED BY "stuffedpoodle";
```

Tada! You now have your own user account. Let's say you chose **ted** as your username. You've configured MySQL to say that **ted** can play around with the **people** database in whatever ways he wishes.

Now get out of MySQL by typing

```
exit
```

To start working with the **people** database, you can now login as **ted**:

```
mysql -u ted -p
```

Creating tables with information in your database:

In MySQL information is stored in tables. Tables contain columns and rows.

Ted has now created a **people** database. So we want now to enter some information into a table.

Login as **ted**.

Firstly, we need to make sure we're working with the **people** database. So typing:

```
select database();
```

will show you what database your currently using. You should see a **NULL** , meaning that your working with nothing at the moment.

So to start using the people database, type:

```
\u people
```

(NOTICE: Typing: **USE people** OR logging in as **mysql people -u ted -p** is also acceptable.)

So how to create a table.

Keep in mind that we need to set all the column values (like surname, age etc.).

Now, remember that annoying -> symbol? MySQL reads your command as just one command, not a series. So, -> enables you to enter your inputs in a nicer way than just writing everything on one line. (NOTE: The problem with this method is that if you screw up on a line and press ENTER to go to the next line, you can't go back and fix your mistake. That's why a nice way to do this is using something like *SciTE Text Editor* (set language to **SQL**) to write your code and just copy/paste that into the shell.)

Another thing is that you must separate your lines with **,** at the end of each line except when you've written your two last lines. On the second to last line, **don't** add **,** and the last line always ends with **;** .

First I have to explain a few things so you're not blown away by an unfamiliar bunch of code.

If you don't know, we use brackets **()** to **encapsulate** code. (Often called *parenthesis*).

The first thing we will be writing after the **CREATE TABLE *tableName*** and the first bracket will be the *database ID* number(we use integers [5] (http://en.wikipedia.org/wiki/Integer)) of each person, mainly known as the **Primary Key**. It's kinda like a passport ID number. Each number is unique to its owner and it has to be to prevent duplication and imposters.

Now, any variable in SQL is created as

```
variableNAME variableTYPE otherVariableAttributes
```

. So in order to **define** the Primary Key variable, we need to type for example:

peopleID(variableNAME) **int**(variableTYPE - short for "integer") **unsigned**(means we want our integer value to always be a positive number) **not null**(we want each row to have a value, so obviously the value can't be empty(NULL)) **auto_increment**(this ensures that each new row that is created will be a unique value) **primary key**(we are saying that this particular variable will be our Primary Key for this Table.)**,** (a reminder that the **,** symbol indicates the end of this line so MySQL knows to go to the next line)

You already know about the **int** variable. There is another which is kinda like *String* (for example: if you've programmed in Java before). It's called **varchar** which stands for *variable characters.* You set the amount of characters someone is able to input into a **varchar** variable. Like this: **nameOfFattestMooseAlive varchar(30)** So **nameOfFattestMooseAlive** can have a maximum of 30 characters.

Okay, so let's see an example of how to create a table relating to the **people** database:

```
CREATE TABLE peopleInfo
(
peopleID int unsigned not null auto_increment primary key,
firstName varchar(30),
lastName varchar(30),
age int,
gender varchar(13)
);
```

Just a note that I set the maximum value of **gender** to 13 because "hermaphrodite" has 13 characters. :)

Now you can type: **CREATE TABLE peopleInfo** and press ENTER if you'd like to start -> and write the rest of the code or you can use SCITE and copy/paste it into your shell.

Great. We now completed our first Table.

Now comes the part when we have to get some actual people into our **peopleInfo** Table.

Since your already using the **people** database, you can type

```
show tables;
```

to see what tables are currently in your database. To see the *properties* of your table type:

```
describe peopleInfo;
```

So, how to fill in our **peopleInfo** table with people...

This is done by telling MySQL **what** *rows* you are filling in and the **actual information/data** you want to fill in.

So we want to **insert into** our table (specifying the rows) and inputting the **values**(actual data) that we want. (NOTE: We are not filling in the primary key.)

To create our first person you would type this:

```
INSERT INTO peopleInfo
(firstName, lastName, age, gender)
values
("Bill", "Harper", 17, "male");
```

Great. Now if you want to printout to the screen all the information about your table, type:

```
select * from peopleInfo;
```

and there you have it. Your table now has one person stored in it.

Inserting lots of information into your table:

A brief point that shall be covered later, MySQL backs-up itself in .sql files. The reason this is smart is because it backs-up the actual code inside the text file.

Keeping this in mind, let's say we want to add 10 other people into your peopleInfo table. It would be one hell of a hassle typing each person into existence. What if there were a 1000?

So I've graciously typed out the code of filling in 10 other people to a database. :) Create a blank .txt file and copy/paste this information into it, saving it as **tenPeople.sql** .

```
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Mary", "Jones", 21, "female");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Jill", "Harrington", 19, "female");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Bob", "Mill", 26, "male");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Alfred", "Jinks", 23, "male");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Sandra", "Tussel", 31, "female");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Mike", "Habraha", 45, "male");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("John", "Murry", 22, "male");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Jake", "Mechowsky", 34, "male");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Hobrah", "Hinbrah", 24, "hermaphrodite");
INSERT INTO peopleInfo (firstName, lastName, age, gender) values ("Laura", "Smith", 17, "female");
```

Excellent. Now we want to get all these people in our table. **exit** MySQL and go to the directory where you saved the **tenPeople.sql** file.

Once there, to get all the the data into your database, type:

```
mysql -u ted -p people <tenPeople.sql
```

and enter your password.

Now log into MySQL and remember to select the database your using. **\u people**

Now check again what information you have. There ya go.

Manipulating your database:

Now that we have a database full of people. We can display that information anyway we want.

A brief example would be

```
select firstName, lastName, gender from peopleInfo;
```

This would display to the screen only peoples name, surname and gender. You've not specified that you want peoples Database ID Number or Age to be displayed. And the great thing is you can choose whatever you want from the database to be displayed. Now, if you want to delete your table, simply type:

```
drop table peopleInfo;
```

Extra conditions:

You can also you extra conditions (filters) through when displaying data.

```
select * from peopleInfo where gender = 'female';
```

will display everyone who is female.

(NOTE: letters are enclosed with ' while numbers are plain.)

You can also compare numbers. For example:

```
select * from peopleInfo where age > 17;
```

will show everyone in your table who is older than 17.

Little index here:

```
>   greater than
<   less than
>=  greater or equal to
<=  less than or equal to
<>  not equal to
```

Let's say we wanted to display all people whose first names began with the letter "j". We would use the LIKE condition. (Makes sense, is your name LIKE the letter "j", well it start with j so yes. :))

About the LIKE condition.

[6] (http://www.webdevelopersnotes.com/tutorials/sql/mysql_reference_guide_pattern_matching_with_text_data.php3)

```
select * from peopleInfo where firstName LIKE "j%";
```

(NOTE: LIKE 's evil opposite cousin is NOT LIKE)

Backing up and restoring your MySQL database:

There is a function called **mysqldump**. This is a way to backup your database.

Remember how you managed to get information into your database from **tenPeople.sql**? Well that's how you restore information to a database.

(In this particular case you gotta make sure that in your database you have a table called "peopleInfo")

Now...

To backup your database (in this case backup the **people** database):

We first have to create the .txt file that we will be backing it up to. Open a blank .txt file and save it as **backupfile.sql** .

Now we can type:

```
mysqldump -u ted -p people > backupfile.sql
```

Congratulations. You have now backuped your people **database.**

WARNING! `mysqldump` is one of the worst ways to backup production databases for the following reasons:

- it will take quite a lot of time to dump data
- even more time to restore. Depends on datasize, it can be counted in days!
- locking problem with MyISAM tables or mixed environment

Better solutions are based on binary copy. It allows you to perform non-locking, consistent backups.

For MyISAM or mixed environment:

- LVM snapshots

For InnoDB:

- LVM snapshots
- ZFS snapshots (for Solaris systems)
- InnoDB Hot Backup
- XtraBackup (similar to InnoDB Hot Backup but free)

phpMyAdmin

This graphic interface allows the generation of SQL code by selecting some options with the mouse. This software has its own wiki on http://wiki.cihar.com /pma/Welcome_to_phpMyAdmin_Wiki.

Hello world

To enter the SQL commands:

- Launch MySQL in shell:
  - Linux: `mysql` -h localhost -u root MyDB
  - Windows: `"C:\Program Files (x86)\EasyPHP\binaries\mysql\bin\mysql.exe"` -h localhost -u root MyDB
- Or open an SQL window in PhpMyAdmin (eg: http://localhost/modules/phpmyadmin/#PMAURL-1:server_sql.php?server=1).

```
select "hello world";
+-------------+
| hello world |
+-------------+
| hello world |
+-------------+
1 row in set (0.00 sec)
```

Databases manipulation

Creation

```
CREATE DATABASE database;
```

Require? Privilege.

`mysqladmin create` is a command-line wrapper for this function.

NB: in MySQL, `CREATE SCHEMA` is a perfect synonym of `CREATE DATABASE`, contrarily to some other DBMS like Oracle or SQL Server.

Deletion

```
DROP DATABASE database;
```

Require ? privilege.

`mysqladmin drop` is a command-line wrapper for this function. The `-f` option can be used to suppress the interactive confirmation (useful for unattended scripts).

Rename

In some 5.1.x versions there was a `RENAME DATABASE db1 TO db2;` command, but it has been removed because renaming databases via SQL caused some data loss problems[1].

However, in the command-line, you can create/export/import/delete:

```
mysqladmin create name2
mysqldump --opt name1 | mysql name2
mysqladmin drop -f name1
```

Another option, if you have root access, is to rename the database directory:

```
cd /var/lib/mysql/
/etc/init.d/mysql stop
mv name1/ name2/
/etc/init.d/mysql start
```

You also need to drop privileges on name1 and recreate them on name2:

```
UPDATE mysql.db SET `Db`='name2' WHERE `Db`='name1';
FLUSH PRIVILEGES;
```

Copy

There is no direct copy command in MySQL. However, this can easily be done using some tools.

With mysqldump

The mysqldump command-line can be used to generate a complete flat-file copy of the database. You can then reinject this copy in another database.

This requires a direct access to the database; if you do not have it, you may need to use phpMyAdmin instead.

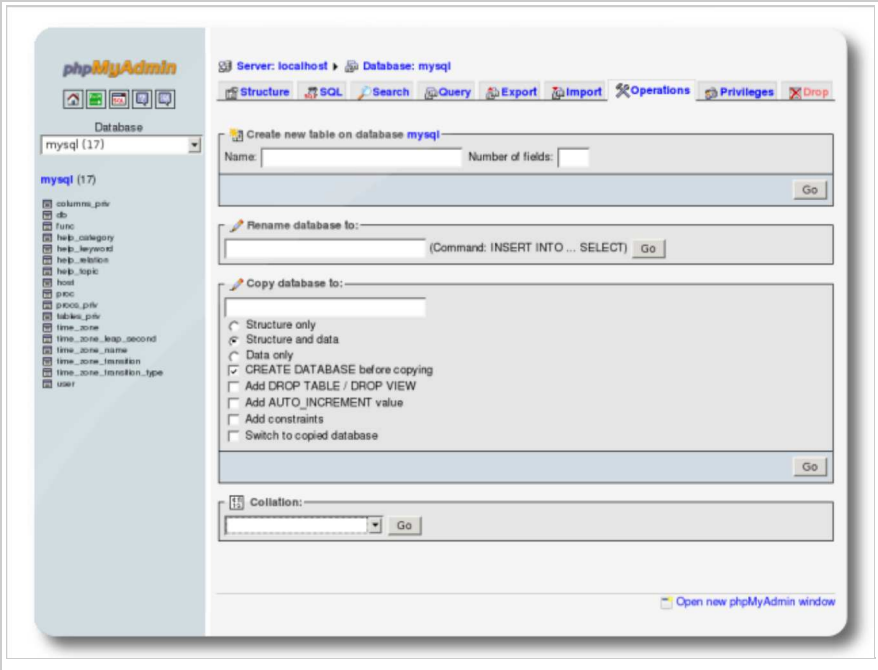
```
# First, clean-up the target database:
mysqladmin drop -f base2
mysqladmin create base2
# Copy base1 to base2:
mysqldump --opt base1 | mysql base2
```

Backup

To set an automatic backup every day at midnight[2], in Linux:

```
$ crontab -e
0 0 * * * /usr/local/bin/mysqldump -uLOGIN -PPORT -hHOST -pPASS base1 | gzip -c > `date "+\%Y-\%m-\%d"`.gz
```

With phpMyAdmin



Restoration

- With Linux:

```
mysql -h localhost -u root MaBase < MaBase.sql
```

- With Windows, the program may not be into the environment variables:

```
"C:\Program Files (x86)\EasyPHP\binaries\mysql\bin\mysql.exe" -h localhost -u root MyDB < MyDB.sql
```

Contrarily to the PhpMyAdmin importations, there is no limit. For example, we can load a 2 GB database in five minutes.

Migration from other databases

Tools: MySQL Migration Toolkit (http://mysql.com/products/tools/migration-toolkit/)

Tools for data modeling

- MySQL Query Browser apparently includes a *MySQL Table Editor* module.
- Kexi (http://www.kexi-project.org/) (wikipedia: Kexi)

DB Designer 4 and MySQL Workbench

DBDesigner begins to be old. It is released under the GNU GPL, but it cannot be fully considered as free software since it requires the non-free Kylix compiler to build.

But MySQL AB acquired fabFORCE[*citation needed*][3], who distributed DB Designer, and MySQL Workbench is the next version. For now the project is still Alpha and not ready for use yet.

Meanwhile, if you use the latest release of DBDesigner, you'll find that it cannot connect to MySQL, with the "unable to load libmysqlclient.so" error. To workaround this,

- Install the MySQL "Shared compatibility libraries" (from http://dev.mysql.com/downloads/mysql/5.0.html#downloads for version 5.0, generic RPMS aka MySQL-shared-compat.i386 will do).
- Replace DBDesigner's version of libmysqlclient.so with the newly installed one:

```
sudo ln -sf /usr/lib/libmysqlclient.so.10 /usr/lib/DBDesigner4/libmysqlclient.so
```

- Find and install `kylixlibs3-unwind-3.0-rh.4.i386.rpm`
- Find an old xorg (e.g. `xorg-x11-libs-6.8.2-37.FC4.49.2.1.i386.rpm` from FC4) and extract it:

```
rpm2cpio x.rpm | cpio -i
```

- Get libXft.so.1.1 in that package and install it:

```
sudo cp libXft.so.1.1 /usr/lib
ldconfig
```

You now can connect to your MySQL5 server from DBDesigner4. Consider this a temporary work-around waiting for community (free) and commercial (not free) versions MySQL Workbench.

OpenOffice Base and ODBC

Typical configuration :

- MySQL database on a host machine (which name is `mysqlhost` below)
- OOo 2 on a client machine (Debian GNU/Linux for instance)
- Connection via ODBC (http://en.wikipedia.org/wiki/ODBC).

It's a client configuration : we need `mysql-client`:

```
aptitude install mysql-client
```

Under Fedora/CentOS:

```
yum install mysql
```

Before installing ODBC, we can test the remote connexion locally:

```
$ mysql -h mysqlhost -u user1 mysqldatabase -p
Enter password: PassUser1
```

You must have create the database `mysqldatabase` and the user `user1` on `mysqlhost`. It seems there is no problem (hope there is not ;-)):

```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 33 to server version: 5.0.24a-Debian_5~bpo.1-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Then, it's possible to test, through different queries :

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysqldatabase      |
+--------------------+
2 rows in set (0.00 sec)
....
mysql> quit;
Bye
```

Fine ! Let's go with OOo and ODBC, on the client machine:

```
aptitude install libmyodbc unixodbc
```

For Fedora/CentOS:

```
yum install mysql-connector-odbc unixODBC
```

`/etc/odbc.ini` (empty file) and `/etc/odbcinst.ini` are created. `odbcinst.ini` declares the available ODBC driver. Here's the MySQL statement (paths to the .so files may vary depending on the distribution); for Debian:

```
[MySQL]
Description     = MySQL driver
Driver          = /usr/lib/odbc/libmyodbc.so
Setup           = /usr/lib/odbc/libodbcmyS.so
CPTimeout       =
CPReuse         =
FileUsage       = 1
```

for CentOS:

```
[MySQL]
Description     = ODBC for MySQL
Driver          = /usr/lib/libmyodbc3.so
Setup           = /usr/lib/libodbcmyS.so
FileUsage       = 1
```

Now we can use `odbcinst` :

```
# odbcinst -j
unixODBC 2.2.4
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
USER DATA SOURCES..: /root/.odbc.ini
```

For further options : `man odbcinst`

First of all, we have to create at least one DSN (Data Source Name or Data Set Name), because every ODBC connection is initialized through an existing DSN. It's true in every cases, so it is required for an ODBC connection from OOo.

To create a DSN, one have different possibilities :

- Modify /etc/odbc.ini (concerns all users)
- Modify ~/.odbc.ini (concerns a specific user)
- Use graphical applications such as **ODBCConfig** (Debian: `unixodbc-bin`, Fedora: `unixODBC-kde`). Finally, these graphical applications modify **/etc/odbc.ini** or **~/.odbc.ini**

For instance, a `/etc/odbc.ini` file (the name of the DSN is between brackets []):

```
[MySQL-test]
Description     =       MySQL ODBC Database
TraceFile       =       stderr
Driver          =       MySQL
SERVER          =       mysqlhost
USER            =       user1
PASSWORD        =
DATABASE        =       mysqldatabase
```

In that case, the DSN is called **MySQL-test**

Then we can test, using **isql** command:

```
$ isql -v MySQL-test user1 PassUser1
+---------------------------------------+
| Connected!                            |
|                                       |
| sql-statement                         |
| help [tablename]                      |
| quit                                  |
|                                       |
+---------------------------------------+
SQL> show databases;
+------------------+
| Database         |
+------------------+
| information_schema|
| mysqldatabase    |
+------------------+
2 rows affected
2 rows returned
SQL> quit;
```

And now, from OOo:

```
-> File
 -> New
  -> Database
-> Connecting to an existing database
 -> MySQL
   -> Next
-> Connect using ODBC
 -> Next
-> Choosing a Data Source
 -> MySQL-test
  -> Next
-> Username : user1 (tick password required)
-> Yes, register the database for me
-> Finish
```

At that step, one is connected to the **mysqldatabase** database, under the user **user1**. Just before accessing the database, for example to create tables, one will give user1 password. Then, through OOo, it is now quite easy to access and manipulate the database. We can just notice that Java is required in the following cases :

- Wizard to create a form (at the opposite, to create a form directly don't need any JRE).
- Wizard to create reports.
- Wizard to create queries (at the opposite, to create a query directly or through a view don't need any JRE).
- Wizard to create tables (at the opposite, to create a table directly or to create a view don't need any JRE).

GNU/Linux distros usually ships OpenOffice with IcedTea (`openjdk-6-jre/java-1.6.0-openjdk`) or GCJ (`java-gcj-compat/java-1.4.2-gcj-compat`) so that these Java-based features work.

References

1. https://dev.mysql.com/doc/refman/5.1/en/rename-database.html
2. http://stackoverflow.com/questions/6645818/how-to-automate-database-backup-using-phpmyadmin
3. In the forums: [1] (http://www.mysqltalk.org/db-designer-4-vt146168.html) but we'd need something more official

Language

Language/Browsing the databases

INFORMATION_SCHEMA

information_schema database into phpMyAdmin.

`information_schema` is a virtual database provided by MySQL 5 and later, that contains metadata about the server and the databases.

You can't modify structure and data of `information_schema`. You can only query the tables.

Many `information_schema` tables provide the same data you can retrieve with a SHOW statement. While using SHOW commands is faster (the server responds much faster and you type less characters), the `information_schema` provides a more flexible way to obtain and organize the metadata.

List databases

The INFORMATION_SCHEMA table containing the databases information is SCHEMATA.

The `mysqlshow` command line tool (DOS/Unix) can be used instead. You can't show databases if the server has been started with the --skip-all-databases option.

If you don't have the 'SHOW DATABASES' privilege, you'll only see databases on which you have some permissions.

The following SQL commands provide information about the databases located on the current server.

Show all databases:

```
SHOW DATABASES;
```

The SCHEMA keywords can be used in place of DATABASES. MySQL doesn't support standard SQL SCHEMAs, so SCHEMA is a synonym of database. It has been added for compatibility with other DBMSs.

Add a filter on the databases names

```
SHOW DATABASES LIKE 'pattern';
```

The LIKE operator here works as in normal SELECTs or DML statements. So you can list all databases whose name starts with 'my':

```
SHOW DATABASES LIKE 'my%';
```

Add complex filters

You can add more complex filters using the WHERE clause:

```
SHOW DATABASES WHERE conditions;
```

WHERE clause allows you to use regular expressions, '=', '<' and '>' operators, string functions or other useful expressions to filter the records returned by SHOW DATABASES.

List tables and views

The following SQL commands provide information about the tables and views contained in a database. The INFORMATION_SCHEMA tables containing this information are `TABLES` and `VIEWS`.

Since the following statements provide very little information about views, if you need to get metadata about them you'll probably prefer to query the VIEWS table.

The `mysqlshow` command line tool can be used instead.

Show all tables

```
USE `database`;
SHOW TABLES;

SHOW TABLES FROM `database`;
```

The 2 forms shown above are equivalent.

Apply a filter

You can apply a filter to the tables names, to show only tables whose name match a pattern. You can use the LIKE operators, as you do in SELECTs or in the DML statements:

```
SHOW TABLES LIKE `pattern`;
```

Also, you can apply a more complex filter to any column returned by the SHOW TABLES command using the WHERE clause:

```
SHOW TABLES WHERE condition;
```

(see below)

Extra info

By default, SHOW TABLES returns only one column containing the name of the table. You can get extra information by using the FULL keyword:

```
SHOW FULL TABLES;
```

This will add a column called `Table_type`. This can have three values: 'BASE TABLE' for tables, 'VIEW' for views and 'SYSTEM VIEW' for special tables created by the server (normally used only INFORMATION_SCHEMA tables).

So you can only list tables:

```
SHOW FULL TABLES WHERE `Table_type`='BASE TABLE';
```

Or, you can only list views:

```
SHOW FULL TABLES WHERE `Table_type`='VIEW';
```

Show only open tables

You can get a list of the non-temporary tables (not views) which are open in the cache:

```
SHOW OPEN TABLES;
```

This command has the same parameters as SHOW TABLES, except for FULL (useless in this case). You can't get this information from the INFORMATION_SCHEMA.

List fields

The following SQL commands provide information about the columns in a table or in a view. The INFORMATION_SCHEMA table containing this information is COLUMNS.

The `mysqlshow` command line tool can be used instead.

DESCRIBE

```
DESCRIBE `table`;
DESCRIBE `database`.`table`;
DESCRIBE `table` 'filter';
```

DESC can be used as a shortcut for DESCRIBE.

'filter' can be a column name. If a column name is specified, only that column will be shown. If 'filter' contains the '%' or the '_' characters, it will be evaluated as

a LIKE condition. For example, you can list all fields which start with 'my':

```
DESC `table` 'my%';
```

EXPLAIN

A synonym is:

```
EXPLAIN `table`;
```

SHOW FIELDS

Another synonym is:

```
SHOW FIELDS FROM `table`;
```

SHOW COLUMNS

Another synonym is:

```
SHOW COLUMNS FROM `table`;

-- possible clauses:
SHOW COLUMNS FROM `table` FROM `database`;
SHOW COLUMNS FROM `table` LIKE 'pattern';
SHOW COLUMNS FROM `table` WHERE condition;
```

FIELDS and COLUMNS are synonyms. EXPLAIN is a synonym of SHOW COLUMNS / FIELDS too, but it doesn't support all of its clauses.

A databases name can be specified both in the form

```
SHOW COLUMNS FROM `table` FROM `database`;
```

both:

```
SHOW COLUMNS FROM `database`.`table`;
```

Extra info

Using the FULL keyword, extra info can be retried: the columns' collation, privileges you have on the column and the comment.

Field	Type	Collation	Null	Key	Default	Extra	Privileges	Comment
...	...	...	...	...	...	...	...	...

List indexes

The following SQL commands provide information about the indexes in a table. Information about keys is contained in the `COLUMNS` table in the INFORMATION_SCHEMA.

The `mysqlshow -k` command line tool can be used instead.

```
SHOW INDEX FROM `TABLE`;
SHOW INDEX FROM `TABLE` FROM `databases`;
```

The KEYS reserved word can be used as a synonym of INDEX. No other clauses are provided.

Result example:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
Table1	0	PRIMARY	1	id	A	19	NULL	NULL		BTREE		

Remark: with phpMyAdmin it's easy to create the same index multiple times, which slows the requests.

To remove an index:

```
DROP INDEX `date_2` on `Table1`
```

Language/Specifying names

In this book, we will quote the MySQL identifiers (tables names, fields, databases, etc.) using backquotes (`).

Backquote is ASCII 96. It can be type on Linux systems by pressing: ALT+'.

Most often, this is optional. However, this allows better error messages from MySQL. For example, this error is not very helpful:

```
mysql> SELECT user_id, group_id FROM user,group LIMIT 1;
ERROR 1064 (42000): You have an error in your SQL syntax;
 check the manual that corresponds to your MySQL server version
 for the right syntax to use near 'group LIMIT 1' at line 1
```

But this one is better:

```
mysql> SELECT user_id, group_id FROM `user`,`group` LIMIT 1;
ERROR 1146 (42S02): Table 'savannah.group' doesn't exist
```

Ok, it was just a missing s:

```
mysql>  SELECT user_id, group_id FROM `user`,`groups` LIMIT 1;
+---------+----------+
| user_id | group_id |
+---------+----------+
|     100 |        2 |
+---------+----------+
1 row in set (0.02 sec)
```

This syntax allows the user to use reserved words and some illegal characters in objects' names. It is even possible to use backquotes by typing it twice:

```
RENAME TABLE `user` TO ````
```

However, this is not a portable syntax. The SQL standard recommends the use of a double quote ("). If you want to write portable SQL quote, do not quote the identifiers. But is there something like portable SQL, even remotely?

Language/Definitions: what are DDL, DML and DQL?

- DDL (Data Definition Language) refers to the CREATE, ALTER and DROP statements

DDL allows to add / modify / delete the logical structures which contain the data or which allow users to access / mantain the data (databases, tables, keys, views...). DDL is about "metadata".

- DML (Data Manipulation Language) refers to the INSERT, UPDATE and DELETE statements

DML allows to add / modify / delete data itself.

- DQL (Data Query Language) refers to the SELECT, SHOW and HELP statements (queries)

SELECT is the main DQL instruction. It retrieves data you need. SHOW retrieves infos about the metadata. HELP... is for people who need help.

- DCL (Data Control Language) refers to the GRANT and REVOKE statements

DCL is used to grant / revoke permissions on databases and their contents. DCL is simple, but MySQL's permissions are rather complex. DCL is about security.

- DTL (Data Transaction Language) refers to the START TRANSACTION, SAVEPOINT, COMMIT and ROLLBACK [TO SAVEPOINT] statements

DTL is used to manage transactions (operations which include more instructions none of which can be executed if one of them fails).

Language/User Variables

Local variables

The local variables can't be reached from outside their function or stored procedure[1].

They are declared like this[2]:

```
DECLARE MyVariable1 INT DEFAULT 1;
```

Session variables

- The ability to set variables in a statement with the := assignment operator:

- For e.g. (@total) to calculate the total in an example, you have to have the total column first because it must be calculated before the individual percentage calculations.
- Session variables are set for the duration of the thread.
- In the vast majority of cases you'd use a programming language to do this sort of thing.
- MySQL variables can be useful when working on the MySQL command line.
- If no records are returned, the user variable will not be set for that statement.
- A user variable set in the field list cannot be used as a condition.
- The value of a variable is set with the SET statement or in a SELECT statement with :=

```
select @test := 2;
select @test + 1;  -- returns 3

set @startdate='some_start_date', @enddate='some_end_date'

SELECT @toremember:=count(*) FROM membros;

select @numzero := count(*) from table1 where field=0;
select @numdistinct := count(distinct field) from table1 where field <> 0 ;
select @numzero @numdistinct;
```

- You can copy values retrieved by a SELECT into one or more variables:

```
SET @id = 0, @name = '';
SELECT id, name INTO @id, @name FROM table1 limit 1;
SELECT @id, @name;
```

Global variables

A global variable is visible to all users, it allows to modify the configuration files settings during the session or definitely. So when changing them, it's necessary to precise this permanent or ephemera criteria, with respectively *set global* and *set session*. Example:

```
mysql> set @@global.max_connections = 1000;
mysql> show global variables like 'wait_timeout';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| wait_timeout | 60 |
+---------------+-------+
1 row in set (0.00 sec)
mysql> set @@session.wait_timeout=120;
```

References

1. http://stackoverflow.com/questions/1009954/mysql-variable-vs-variable-whats-the-difference
2. http://dev.mysql.com/doc/refman/5.7/en/declare-local-variable.html

Language/Alias

An expression and a column may be given aliases using AS. The alias is used as the expression's column name and can be used with order by or having clauses. For e.g.

```
SELECT
    CONCAT(last_name,' ', first_name) AS full_name,
    nickname AS nick
FROM
    mytable
ORDER BY
    full_name
```

These aliases can be used in ORDER BY, GROUP BY and HAVING clauses. They should not be used in WHERE clause.

A table name can have a shorter name for reference using AS. You can omit the AS word and still use aliasing. For e.g.

```
SELECT
    COUNT(B.Booking_ID), U.User_Location
FROM
    Users U
LEFT OUTER JOIN
    Bookings AS B
ON
    U.User_ID    = B.Rep_ID AND
    B.Project_ID = '10'
GROUP BY
    (U.User_Location)
```

Aliasing plays a crucial role while you are using self joins. For e.g. people table has been referred to as p and c aliases!

```
SELECT
    p.name                                AS parent,
    c.name                                AS child,
    MIN((TO_DAYS(NOW())-TO_DAYS(c.dob))/365) AS minage
```

```
FROM
    people AS p
LEFT JOIN
    people AS c
ON
    p.name=c.parent WHERE c.name IS NOT NULL
GROUP BY
    parent HAVING minage > 50 ORDER BY p.dob;
```

Language/Data Types

VARCHAR

VARCHAR is shorthand for CHARACTER VARYING. 'n' represents the maximum column length (upto 65,535 characters). A VARCHAR(10) column can hold a string with a maximum length of 10 characters. The actual storage required is the length of the string (L), plus 1 or 2 bytes (1 if the length is < 255) to record the length of the string.
For the string 'abcd', L is 4 and the storage requirement is 5 bytes.

CHAR(n) is similar to varchar(n) with the only difference that char will occupy fixed length of space in the database whereas varchar will need the space to store the actual text.

TEXT and BLOB

A BLOB or TEXT column with a maximum length of 65,535 characters. The required space is the real length of the stored data plus 2 bytes (1 byte if length is < 255). The BLOB / TEXT data is not stored in the table's datafile. This makes all operations (INSERT / UPDATE / DELETE / SELECT) involving the BLOB / TEXT data slower, but makes all other operations faster.

integer

Specifying an n value has no effect whatsoever. Regardless of a supplied value for n, maximum (unsigned) value stored is 429 crores. If you want to add negative numbers, add the "signed" keyword next to it.

decimal

decimal(n,m) decimal(4,2) means numbers upto 99.99 (and NOT 9999.99 as you may expect) can be saved. Four digits with the last 2 reserved for decimal.

Dates

Out of the three types DATETIME, DATE, and TIMESTAMP, the DATE type is used when you need only a date value, without a time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The DATETIME type is used when you need values that contain both date and time information. The difference between DATETIME and TIMESTAMP is that the TIMESTAMP range is limited to 1970-2037 (see below).

TIME can be used to only store the time of day (HH:MM:SS), without the date. It can also be used to represent a time interval (for example: -02:00:00 for "two hours in the past"). Range: '-838:59:59' => '838:59:59'.

YEAR can be used to store the year number only.

If you manipulate dates, you have to specify the actual date, not only the time - that is, MySQL will not automagically use today as the current date. On the contrary, MySQL will even interpret the HH:MM:SS time as a YY:MM:DD value, which will probably be invalid.

The following examples show the precise date range for Unix-based timestamps, which starts at the Unix Epoch and stops just before the first new year before the $2^{31} - 1$ usual limit (2038).

```
mysql> SET time_zone = '+00:00'; -- GMT
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT FROM_UNIXTIME(-1);
+-------------------+
| FROM_UNIXTIME(-1) |
+-------------------+
| NULL              |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT FROM_UNIXTIME(0); -- "Epoch"
+---------------------+
| FROM_UNIXTIME(0)    |
+---------------------+
| 1970-01-01 00:00:00 |
+---------------------+
1 row in set (0.00 sec)

mysql> SELECT FROM_UNIXTIME(2145916799);
+---------------------------+
| FROM_UNIXTIME(2145916799) |
+---------------------------+
| 2037-12-31 23:59:59       |
+---------------------------+
```

```
1 row in set (0.00 sec)

mysql> SELECT FROM_UNIXTIME(2145916800);
+---------------------------+
| FROM_UNIXTIME(2145916800) |
+---------------------------+
| NULL                      |
+---------------------------+
1 row in set (0.00 sec)
```

set and enum

A SET datatype can hold any number of strings from a predefined list of strings specified during table creation. The SET datatype is similar to the ENUM datatype in that they both work with predefined sets of strings, but where the ENUM datatype restricts you to a single member of the set of predefined strings, the SET datatype allows you to store any of the values together, from none to all of them.

Example:

```
SET("madam", "mister")    -- authorizes an empty field, "madam", "mister", "madam, mister", or "mister, madam"

ENUM("madam", "mister")   -- authorizes an empty field, "madam" or  "mister"
```

Language/Table manipulation

CREATE TABLE

Create table syntax is:

```
Create table tablename (FieldName1 DataType, FieldName2 DataType)
```

The rows returned by the "select" query can be saved as a new table. The datatype will be the same as the old table. For e.g.

```
CREATE TABLE LearnHindi
select english.tag, english.Inenglish as english, hindi.Inhindi as hindi
FROM english, hindi
WHERE english.tag = hindi.tag
```

The table size limit depends on the filesystem, and is generally around 2TB[1].

Temporary tables

It's possible to create variables of type "table", which as the other variables, will be erased at the end of their scripts. It's called the "temporary tables":

```
CREATE TEMPORARY TABLE IF NOT EXISTS MyTempTable1 AS (SELECT * FROM MyTable1)
```

Example with a named column:

```
CREATE TEMPORARY TABLE IF NOT EXISTS MyTempTable1(id INT) AS (SELECT id FROM MyTable1)
```

Attention: if the temporary table column name doesn't correspond to the field which fills it, an additional column will be added with this field name. Eg:

```
CREATE TEMPORARY TABLE IF NOT EXISTS MyTempTable1(id1 INT) AS (SELECT id FROM MyTable1);
SHOW FIELDS FROM MyTempTable1;
```

```
Field      Type    Null    Key     Default         Extra
id1     int(11)         YES             NULL
id      int(11)         NO              0
```

}}

ALTER TABLE

ALTER TABLE command can be used when you want to add/delete/modify the columns and/or the indexes; or, it can be used to change other table properties.

Add a column:

```
ALTER TABLE awards
ADD COLUMN AwardCode int(2)
```

Modify a column:

```
ALTER TABLE awards
```

```
CHANGE COLUMN AwardCode VARCHAR(2) NOT NULL

ALTER TABLE awards
MODIFY COLUMN AwardCode VARCHAR(2) NOT NULL
```

Drop a column:

```
ALTER TABLE awards
DROP COLUMN AwardCode
```

Re-order the record in a table:

```
ALTER TABLE awards ORDER BY id
```

(this operation is only supported by some Storage Engines; it could make some query faster)

Rename a table

In order to rename a table, you must have ALTER and DROP privileges on the old table name (or on all the tables), and CREATE and INSERT privileges on the new table name (or on all the tables).

You can use ALTER TABLE to rename a table:

```
RENAME TABLE `old_name` TO `new_name`
```

You can rename more than one table with a single command:

```
RENAME TABLE `old1` TO `new1`, `old2` TO `new2`, ...
```

RENAME is a shortcut. You can also use the ALTER TABLE statement:

```
ALTER TABLE `old` RENAME `new`
```

Using ALTER TABLE you can only rename one table per statement, but it's the only way to rename temporary tables.

DROP TABLE

```
DROP TABLE `awards`
```

Will completely delete the table and all the records it contains.

You can also drop more than one table with a single statement:

```
DROP TABLE `table1`, `table2`, ...
```

There are come optional keywords:

```
DROP TEMPORARY TABLE `table`;
DROP TABLE `table` IF EXISTS;
```

TEMPORARY must be specified, to drop a temporary table. IF EXISTS tells the server that it must not raise an error if the table doesn't exist.

References

1. http://dev.mysql.com/doc/refman/5.7/en/table-size-limit.html

Language/Data manipulation

INSERT

The syntax is as follows:

Insert value1 into Column1, value2 into Column2, and value3 into Column3:

```
INSERT INTO TableName (Column1, Column2, Column3)
VALUES (value1, value2, value3)
```

Insert one record (values are inserted in the order that the columns appear in the database):

```
INSERT INTO TableName
VALUES (value1, value2, value3)
```

Insert two records:

```
INSERT INTO TableName
VALUES (value1, value2, value3), (value4, value5, value6)

INSERT INTO antiques VALUES (21, 01, 'Ottoman', 200.00);
INSERT INTO antiques (buyerid, sellerid, item) VALUES (01, 21, 'Ottoman');
```

You can also insert records 'selected' from other table.

```
INSERT INTO table1(field1, field2)
SELECT field1, field2
FROM table2

INSERT INTO World_Events SELECT * FROM National_Events
```

Performance tips:

- To insert many rows, consider using LOAD DATA INFILE instead.
- If bulk INSERTs are too slow and they operate on indexed non-empty tables, maybe you should increase the value of bulk_insert_buffer_size.
- Before performing bulk inserts, you may want to disable the keys.
- LOCKing a table also speeds up the INSERT.

UPDATE

The syntax is:

```
UPDATE table SET field1 = newvalue1, field2 = newvalue2 WHERE criteria ORDER BY field LIMIT n
```

Examples are:

```
UPDATE owner SET ownerfirstname = 'John'
  WHERE ownerid = (SELECT buyerid FROM antiques WHERE item = 'Bookcase');

UPDATE antiques SET price = 500.00 WHERE item = 'Chair';

UPDATE order SET discount=discount * 1.05

UPDATE tbl1 JOIN tbl2 ON tbl1.ID = tbl2.ID
  SET tbl1.col1 = tbl1.col1 + 1
  WHERE tbl2.status='Active'

UPDATE tbl SET names = REPLACE(names, 'aaa', 'zzz')

UPDATE products_categories AS pc
  INNER JOIN products AS p ON pc.prod_id = p.id
  SET pc.prod_sequential_id = p.sequential_id

UPDATE table_name SET col_name =
  REPLACE(col_name, 'host.domain.com', 'host2.domain.com')

UPDATE posts SET deleted=True
  ORDER BY date LIMIT 1
```

With ORDER BY you can order the rows before updating them, and only update a given number of rows (LIMIT).

It is currently not possible to update a table while performing a subquery on the same table. For example, if I want to reset a password I forgot in SPIP:

```
mysql> UPDATE spip_auteurs SET pass =
 (SELECT pass FROM spip_auteurs WHERE login='paul') where login='admin';
ERROR 1093 (HY000): You can't specify target table 'spip_auteurs' for update in FROM clause
```

TODO: [7] (http://www.xaprb.com/blog/2006/06/23/how-to-select-from-an-update-target-in-mysql/) describes a work-around that I couldn't make to work with MySQL 4.1. Currently the work-around is not use 2 subqueries, possibly with transactions.

Performance tips

- UPDATEs speed depends of how many indexes are updated.
- If you UPDATE a MyISAM table which uses dynamic format, if you make rows larger they could be splitted in more than one part. This causes reading overhead. So, if your applications often do this, you may want to regularly run an OPTIMIZE TABLE statement.
- Performing many UPDATEs all together on a LOCKed table is faster than performing them individually.

REPLACE

REPLACE works exactly like INSERT, except that if an old record in the table has the same value as a new record for a PRIMARY KEY or a UNIQUE index, the old record is deleted before the new record is inserted.

With IGNORE, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort.

Prior to MySQL 4.0.1, INSERT ... SELECT implicitly operates in IGNORE mode. As of MySQL 4.0.1, specify IGNORE explicitly to ignore records that would cause duplicate-key violations.

DELETE and TRUNCATE

```
DELETE [QUICK] FROM `table1`
TRUNCATE [TABLE] `table1`
```

- If you don't use a WHERE clause with DELETE, all records will be deleted.
- It can be very slow in a large table, especially if the table has many indexes.
- If the table has many indexes, you can make the cache larger to try making the DELETE faster (key_buffer_size variable).
- For indexed MyISAM tables, in some cases DELETEs are faster if you specify the QUICK keyword (DELETE QUICK FROM ...). This is only useful for tables where DELETEd index values will be reused.
- TRUNCATE will delete all rows quickly by DROPping and reCREATE-ing the table (not all Storage Engines support this operation).
- TRUNCATE is not transaction-safe nor lock-safe.
- DELETE informs you how many rows have been removed, but TRUNCATE doesn't.
- After DELETing many rows (about 30%), an OPTIMIZE TABLE command should make next statements faster.
- For a InnoDB table with FOREIGN KEYs constraints, TRUNCATE behaves like DELETE.

```
DELETE FROM `antiques`
  WHERE item = 'Ottoman'
  ORDER BY `id`
  LIMIT 1
```

You can order the rows before deleting them, and then delete only a given number of rows.

Language/Queries

SELECT

select syntax is as follows:

```
SELECT *
FROM a_table_name
WHERE condition
GROUP BY grouped_field
HAVING group_name condition
ORDER BY ordered_field
LIMIT limit_number, offset
```

List of fields

You must specify what data you're going to retrieve in the SELECT clause:

```
SELECT DATABASE() -- returns the current db's name
SELECT CURRENT_USER() -- returns your username
SELECT 1+1 -- returns 2
```

Any SQL expression is allowed here.

You can also retrieve all fields from a table:

```
SELECT * FROM `stats`
```

If you SELECT only the necessary fields, the query will be faster.

The table's name

If you are retrieving results from a table or a view, usually you specify the table's name in the FROM clause:

```
SELECT id FROM `stats` -- retrieve a field called id from a table called stats
```

Or:

```
SELECT MAX(id) FROM `stats`
SELECT id*2 FROM `stats`
```

You can also use the `db_name`.`table_name` syntax:

```
SELECT id FROM `sitedb`.`stats`
```

But you can also specify the table's name in the SELECT clause:

```
SELECT `stats`.`id` -- retrieve a field called id from a table
```

```
SELECT `sitedb`.`stats`.`id`
```

WHERE

You can set a filter to decide what records must be retrieved.

For example, you can retrieve only the record which has an id of 42:

```
SELECT * FROM `stats` WHERE `id`=42
```

Or you can read more than one record:

```
SELECT * FROM `antiques` WHERE buyerid IS NOT NULL
```

GROUP BY

You can group all records by one or more fields. The record which have the same value for that field will be grouped in one computed record. You can only select the grouped record and the result of some aggregate functions, which will be computed on all records of each group.

For example, the following will group all records in the table `users` by the field `city`. For each group of users living in the same city, the maximum age, the minimum age and the average age will be returned:

```
SELECT city, MAX(age), MIN(age), AVG(age) GROUP BY `city`
```

In the following example, the users are grouped by city and sex, so that we'll know the max, min and avg age of male/female users in each city:

```
SELECT city, sex, MAX(age), MIN(age), AVG(age) GROUP BY `city`, `sex`
```

HAVING

The HAVING clause declares a filter for the records which are computed by the GROUP BY clause. It's different from the WHERE clause, that operates before the GROUP BY. Here's what happens:

1. The records which match to the WHERE clause are retrieved
2. Those records are used to compute new records as defined in the GROUP BY clause
3. The new records that match to the HAVING conditions are returned

This means which WHERE decides what record are used to compose the new computed records.

HAVING decides what computed records are returned, so it can operate on the results of aggregate functions. HAVING is not optimized and can't use indexes.

Incorrect use of HAVING:

```
SELECT city, sex, MAX(age), MIN(age), AVG(age) GROUP BY `city` HAVING sex='m'
```

This probably gives a wrong results. MAX(age) and other aggregate calculations are made using all values, even if the record's sex value is 'f'. This is hardly the expected result.

Incorrect use of HAVING:

```
SELECT city, sex, MAX(age), MIN(age), AVG(age) GROUP BY `city`, `sex` HAVING sex='m'
```

This is correct and returns the expected results, but the execution of this query is not optimized. The WHERE clause can and should be used, because, so that MySQL doesn't computes records which are excluded later.

Correct use of HAVING:

```
SELECT city, sex, MAX(age), MIN(age), AVG(age) GROUP BY `city` HAVING MAX(age) > 80
```

It must group all records, because can't decide the max age of each city before the GROUP BY clause is execute. Later, it returns only the record with a MAX(age)>80.

ORDER BY

You can set an arbitrary order for the records you retrieve. The order may be alphabetical or numeric.

```
SELECT * FROM `stats` ORDER BY `id`
```

By default, the order is ASCENDING. You can also specify that the order must be DESCENDING:

```
SELECT * FROM `stats` ORDER BY `id` ASC -- default
SELECT * FROM `stats` ORDER BY `id` DESC -- inverted
```

NULLs values are considered as minor than any other value.

You can also specify the field position, in place of the field name:

```
SELECT `name`, `buyerid` FROM `antiques` ORDER BY 1 -- name
SELECT `name`, `buyerid` FROM `antiques` ORDER BY 2 -- buyerid
SELECT `name`, `buyerid` FROM `antiques` ORDER BY 1 DESC
```

SQL expressions are allowed:

```
SELECT `name` FROM `antiques` ORDER BY REVERSE(`name`)
```

You can retrieve records in a random order:

```
SELECT `name` FROM `antiques` ORDER BY RAND()
```

If a GROUP BY clause is specified, the results are ordered by the fields named in GROUP BY, unless an ORDER BY clause is present. You can even specify in the GROUP BY clause if the order must be ascending or descending:

```
SELECT city, sex, MAX(age) GROUP BY `city` ASC, `sex` DESC
```

If you have a GROUP BY but you don't want the records to be ordered, you can use ORDER BY NULL:

```
SELECT city, sex, MAX(age) GROUP BY `city`, `sex` ORDER BY NULL
```

LIMIT

You can specify the maximum of rows that you want to read:

```
SELECT * FROM `antiques` ORDER BY id LIMIT 10
```

This statement returns a maximum of 10 rows. If there are less than 10 rows, it returns the number of rows found. The limit clause is usually used with ORDER BY.

You can get a given number of random records:

```
SELECT * FROM `antiques` ORDER BY rand() LIMIT 1 -- one random record
SELECT * FROM `antiques` ORDER BY rand() LIMIT 3
```

You can specify how many rows should be skipped before starting to return the records found. The first record is 0, not one:

```
SELECT * FROM `antiques` ORDER BY id LIMIT 10
SELECT * FROM `antiques` ORDER BY id LIMIT 0, 10 -- synonym
```

You can use the LIMIT clause to get the pagination of results:

```
SELECT * FROM `antiques` ORDER BY id LIMIT 0, 10 -- first page
SELECT * FROM `antiques` ORDER BY id LIMIT 10, 10 -- second page
SELECT * FROM `antiques` ORDER BY id LIMIT 20, 10 -- third page
```

Also, the following syntax is acceptable:

```
SELECT * FROM `antiques` ORDER BY id LIMIT 10 OFFSET 10
```

You can use the LIMIT clause to check the syntax of a query without waiting for it to return the results:

```
SELECT ... LIMIT 0
```

Optimization tips:

- SQL_CALC_FOUND_ROWS may speed up a query [1][2]
- LIMIT is particularly useful for SELECTs which use ORDER BY, DISTINCT and GROUP BY, because their calculations don't have to involve all the rows.
- If the query is resolved by the server copying internally the results into a temporary table, LIMIT helps MySQL to calculate how much memory is required by the table.

DISTINCT

The DISTINCT keyword can be used to remove all duplicate rows from the resultset:

```
SELECT DISTINCT * FROM `stats` -- no duplicate rows
SELECT DISTINCTROW * FROM `stats` -- synonym
SELECT ALL * FROM `stats` -- duplicate rows returned (default)
```

You can use it to get the list of all values contained in one field:

```
SELECT DISTINCT `type` FROM `antiques` ORDER BY `type`
```

Or you can use it to get the existing combinations of some values:

```
SELECT DISTINCT `type`, `age` FROM `antiques` ORDER BY `type`
```

If one of the fields you are SELECTing is the PRIMARY KEY or has a UNIQUE index, DISTINCT is useless. Also, it's useless to use DISTINCT in conjunction with the GROUP BY clause.

IN and NOT IN

```
SELECT id
FROM stats
WHERE position IN ('Manager', 'Staff')

SELECT ownerid, 'is in both orders & antiques'
FROM orders, antiques WHERE ownerid = buyerid
UNION
SELECT buyerid, 'is in antiques only'
FROM antiques WHERE buyerid NOT IN (SELECT ownerid FROM orders)
```

EXISTS and ALL

(Compatible: Mysql 4+)

```
SELECT ownerfirstname, ownerlastname
FROM owner
WHERE EXISTS (SELECT * FROM antiques WHERE item = 'chair')

SELECT buyerid, item
FROM antiques
WHERE price = ALL (SELECT price FROM antiques)
```

Optimization hints

There are some hints you may want to give to the server to better optimize the SELECTs. If you give more than one hints, the order of the keywords is important:

```
SELECT [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY] [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT | SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    ...
```

HIGH_PRIORITY

Usually, DML commands (INSERT, DELETE, UPDATE) have higher priority than SELECTs. If you specify HIGH_PRIORITY though, the SELECT will have higher priority than DML statements.

STRAIGHT_JOIN Force MySQL to evaluate the tables of a JOIN in the same order they are named, from the leftmost.

SQL_SMALL_RESULT It's useful only while using DISTINCT or GROUP BY. Tells the optimizer that the query will return few rows.

SQL_BIG_RESULT It's useful only while using DISTINCT or GROUP BY. Tells the optimizer that the query will return a many rows.

SQL_BUFFER_RESULT Force MySQL to copy the result into a temporary table. This is useful to remove LOCKs as soon as possible.

SQL_CACHE Forces MySQL to copy the result into the query cache. Only works if the value of query_cache_type is DEMAND or 2.

SQL_NO_CACHE Tells MySQL not to cache the result. Useful if the query occurs very seldom or if the result often change.

SQL_CALC_FOUND_ROWS Useful if you are using the LIMIT clause. Tells the server to calculate how many rows would have been returned if there were no LIMIT. You can retrieve that number with another query:

```
SELECT SQL_CALC_FOUND_ROWS * FROM `stats` LIMIT 10 OFFSET 100;
SELECT FOUND_ROWS();
```

Index hints

- `USE INDEX`: specifies to research some records preferably by browsing the tables indexes[3].
- `FORCE INDEX`: idem in more restrictive. A table will be browsed without index only if the optimizer doesn't have the choice.
- `IGNORE INDEX`: request to not favor the indexes.

Example:

```
SELECT *
FROM table1 USE INDEX (date)
WHERE date between '20150101' and '20150131'
```

UNION and UNION All

(Compatible: Mysql 4+)

Following query will return all the records from both tables.

```
SELECT * FROM english
UNION ALL
SELECT * FROM hindi
```

`UNION` is the same as `UNION DISTINCT`.
If you type only UNION, then it is considered that you are asking for distinct records. If you want all records, you have to use `UNION ALL`.

```
SELECT word FROM word_table WHERE id = 1
UNION
SELECT word FROM word_table WHERE id = 2

(SELECT magazine FROM pages)
UNION DISTINCT
(SELECT magazine FROM pdflog)
ORDER BY magazine

(SELECT ID_ENTRY FROM table WHERE ID_AGE = 1)
UNION DISTINCT
(SELECT ID_ENTRY FROM table WHERE ID_AGE=2)
```

Joins

The Most important aspect of SQL is its relational features. You can query, compare and calculate two different tables having entirely different structure. Joins and subselects are the two methods to join tables. Both methods of joining tables should give the same results. The natural join is faster on most SQL platforms.

In the following example a student is trying to learn what the numbers are called in hindi.

```
CREATE TABLE english (Tag int, Inenglish varchar(255));
CREATE TABLE hindi (Tag int, Inhindi varchar(255));

INSERT INTO english (Tag, Inenglish) VALUES (1, 'One');
INSERT INTO english (Tag, Inenglish) VALUES (2, 'Two');
INSERT INTO english (Tag, Inenglish) VALUES (3, 'Three');

INSERT INTO hindi (Tag, Inhindi) VALUES (2, 'Do');
INSERT INTO hindi (Tag, Inhindi) VALUES (3, 'Teen');
INSERT INTO hindi (Tag, Inhindi) VALUES (4, 'Char');
```

select * from english

Tag	Inenglish
1	One
2	Two
3	Three

select * from hindi

Tag	Inhindi
2	Do
3	Teen
4	Char

Cartesian join (CROSS JOIN)

A Cartesian join is when you join every row of one table to every row of another table.

```
SELECT * FROM english, hindi
```

It is also called Cross Join and may be written in this way:

```
SELECT * FROM english CROSS JOIN hindi
```

Tag	Inenglish	Tag	Inhindi
1	One	2	Do
2	Two	2	Do
3	Three	2	Do
1	One	3	Teen
2	Two	3	Teen
3	Three	3	Teen
1	One	4	Char
2	Two	4	Char
3	Three	4	Char

Inner Join

```
SELECT hindi.Tag, english.Inenglish, hindi.Inhindi
FROM english, hindi
WHERE english.Tag = hindi.Tag
-- equal
SELECT hindi.Tag, english.Inenglish, hindi.Inhindi
FROM english INNER JOIN hindi ON english.Tag = hindi.Tag
```

Tag	Inenglish	Inhindi

2	Two	Do
3	Three	Teen

You can also write the same query as

```
SELECT hindi.Tag, english.Inenglish, hindi.Inhindi
FROM english INNER JOIN hindi
ON english.Tag = hindi.Tag
```

Natural Joins using "using" (Compatible: MySQL 4+; but changed in MySQL 5) The following statement using "USING" method will display the same results.

```
SELECT hindi.tag, hindi.Inhindi, english.Inenglish
FROM hindi NATURAL JOIN english
USING (Tag)
```

Outer Joins

Tag	Inenglish	Tag	Inhindi
1	One		
2	Two	2	Do
3	Three	3	Teen
		4	Char

LEFT JOIN / LEFT OUTER JOIN

The syntax is as follows:

```
SELECT field1, field2 FROM table1 LEFT JOIN table2 ON field1=field2

SELECT e.Inenglish as English, e.Tag, '--no row--' as Hindi
FROM english AS e LEFT JOIN hindi AS h
ON e.Tag=h.Tag
WHERE h.Inhindi IS NULL
```

```
English  tag   Hindi
One      1     --no row-
```

Right Outer Join

```
SELECT '--no row--' AS English, h.tag, h.Inhindi AS Hindi
FROM english AS e RIGHT JOIN hindi AS h
ON e.Tag=h.Tag
WHERE e.Inenglish IS NULL
```

English tag Hindi --no row-- 4 Char

- Make sure that you have the same name and same data type in both tables.
- The keywords LEFT and RIGHT are not absolute, they only operate within the context of the given statement: we can reverse the order of the tables and reverse the keywords, and the result would be the same.
- If the type of join is not specified as inner or outer then it will be executed as an INNER JOIN.

Full Outer Join

As for v5.1, MySQL does not provide FULL OUTER JOIN. You may emulate it this way:

```
    (SELECT a.*, b*
        FROM tab1 a LEFT JOIN tab2 b
        ON a.id = b.id)
UNION
    (SELECT a.*, b*
        FROM tab1 a RIGHT JOIN tab2 b
        ON a.id = b.id)
```

Multiple joins

It is possible to join more than just two tables:

```
SELECT ... FROM a JOIN (b JOIN c on b.id=c.id) ON a.id=b.id
```

Here is an example from *Savane*:

```
mysql> SELECT group_type.type_id, group_type.name, COUNT(people_job.job_id) AS count
       FROM group_type
        JOIN (groups JOIN people_job ON groups.group_id = people_job.group_id)
        ON group_type.type_id = groups.type
       GROUP BY type_id ORDER BY type_id
+---------+--------------------------------------+-------+
| type_id | name                                 | count |
+---------+--------------------------------------+-------+
|       1 | Official GNU software                |   148 |
```

```
|       2 | non-GNU software and documentation   |   268 |
|       3 | www.gnu.org portion                  |     4 |
|       6 | www.gnu.org translation team         |     5 |
+---------+--------------------------------------+-------+
4 rows in set (0.02 sec)
```

Subqueries

(Compatible: MySQL 4.1 and later)

- SQL subqueries let you use the results of one query as part of another query.
- Subqueries are often natural ways of writing a statement.
- Let you break a query into pieces and assemble it.
- Allow some queries that otherwise can't be constructed. Without using a subquery, you have to do it in two steps.
- Subqueries always appear as part of the WHERE (or HAVING) clause.
- Only one field can be in the subquery SELECT. It means Subquery can only produce a single column of data as its result.
- ORDER BY is not allowed; it would not make sense.
- Usually refer to name of a main table column in the subquery.
- This defines the current row of the main table for which the subquery is being run. This is called an outer reference.

For e.g. If RepOffice= OfficeNbr from Offices table, list the offices where the sales quota for the office exceeds the sum of individual salespersons' quotas

```
SELECT City FROM Offices WHERE Target > ???
```

??? is the sum of the quotas of the salespeople, i.e.

```
SELECT SUM(Quota)
FROM SalesReps
WHERE RepOffice = OfficeNbr
```

We combine these to get

```
SELECT City FROM Offices
WHERE Target > (SELECT SUM(Quota) FROM SalesReps
WHERE RepOffice = OfficeNbr)
```

Display all customers with orders or credit limits > $50,000. Use the DISTINCT word to list the customer just once.

```
SELECT DISTINCT CustNbr
FROM Customers, Orders
WHERE CustNbr = Cust AND (CreditLimit>50000 OR Amt>50000);
```

References

1. http://www.mysqlperformanceblog.com/2007/08/28/to-sql_calc_found_rows-or-not-to-sql_calc_found_rows/
2. http://dev.mysql.com/doc/refman/5.0/en/information-functions.html
3. http://dev.mysql.com/doc/refman/5.7/en/index-hints.html

Resources

- Official MySQL documentation (http://dev.mysql.com/doc)

Language/Using NULL

Null is a special logical value in SQL. Most programming languages have 2 values of logic: True and False. SQL also has NULL which means "Unknown". A NULL value can be set.

NULL is a non-value, so it can be assigned to TEXT columns, INTEGER columns or any other datatype. A column can not contain NULLs only if it has been declared as NOT NULL (see ALTER TABLE).

```
INSERT into Singer
       (F_Name, L_Name, Birth_place, Language)
       values
       ("", "Homer", NULL, "Greek"),
       ("", "Sting", NULL, "English"),
       ("Jonny", "Five", NULL, "Binary");
```

Do not quote the NULL. If you quote a Null then you name the person NULL. For some strange reason, NULLs do not show visually on windows XP in Varchar fields but they do in Fedora's version, so versions of mysql can give different outputs. Here we set the value of Sting and Homer's first name to a zero length string "", because we KNOW they have NO first name, but we KNOW we do not know the place they were born. To check for a NULLs use

```
SELECT * from Singer WHERE Birth_place IS NULL;
or
```

```
SELECT * from Singer WHERE Birth_place IS NOT NULL;
or
SELECT * from Singer WHERE isNull(Birth_place)
```

Remember, COUNT never counts NULLS.

```
select count(Birth_place) from Singer;
0
and sum(NULL) gives a NULL answer.
```

Normal operations (comparisons, expressions...) return NULL if at least one of the compared items is NULL:

```
SELECT (NULL=NULL) OR (NULL<>NULL) OR (NOT NULL) OR (1<NULL) OR (1>NULL) OR (1 + NULL) OR (1 LIKE NULL)
```

because all the expressions between in parenthesis return NULL. It's definitely logical: if you don't know the value represented by NULL, you don't know is it's =1 or <>1. Be aware that even (NULL=NULL and (NOT NULL) return NULL.

Dealing with NULL

The function 'COALESCE' can simplify working with null values. for example, to avoid showing null values by treating null as zero, you can type:

```
SELECT COALESCE(colname,0) from table where COALESCE(colname,0) > 1;
```

In a date field, to treat NULL as the current date:

```
ORDER BY (COALESCE(TO_DAYS(date),TO_DAYS(CURDATE()))-TO_DAYS(CURDATE()))
```

```
EXP(SUM(LOG(COALESCE(''*the field you want to multiply*'',1)))
```

The coalesce() function is there to guard against trying to calculate the logarithm of a null value and may be optional depending on your circumstances.

```
SELECT t4.gene_name, COALESCE(g2d.score,0),
COALESCE(dgp.score,0), COALESCE(pocus.score,0)
FROM t4
LEFT JOIN g2d ON t4.gene_name=g2d.gene_name
LEFT JOIN dgp ON t4.gene_name=dgp.gene_name
LEFT JOIN pocus ON t4.gene_name=pocus.gene_name;
```

Use of IFNULL() in your SELECT statement is to make the NULL any value you wish.

```
IFNULL(expr1,expr2)
```

If expr1 is not NULL, IFNULL() returns expr1, else it returns expr2.

IFNULL() returns a numeric or string value, depending on the context in which it is used:

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

Null handling can be very counter intuitive and could cause problems if you have an incorrect function in a delete statement that returns null. For example the following query will delete all entries.

```
DELETE FROM my_table WHERE field > NULL (or function returning NULL)
```

If you want to have NULL values presented last when doing an ORDER BY, try this:

```
SELECT * FROM my_table ORDER BY ISNULL(field), field [ ASC | DESC ]
```

Language/Operators

MySQL uses some standard SQL operators and some non-standard operators. They can be used to write expressions which involve constant values, variables, values contained in fields and / or other expressions.

Precedence

Operator precedence

Table of operator precedence:

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
&&, AND
XOR
||, OR
:=
```

Modifiers:

- **PIPES_AS_CONCAT** - If this SQL mode is enabled, || has precedence on ^, but - and ~ have precedence on ||.
- **HIGH_NOT_PRECEDENCE** - If this SQL mode is enabled, NOT has the same precedence level as !.

Use of parenthesis

You can use parenthesis to force MySQL to evaluate a subexpression before another independently from operator precedence:

```
SELECT (1 + 1) * 5 -- returns 10
```

You can also use parenthesis to make an expression more readable by humans, even if they don't affect the precedence:

```
SELECT 1 + (2 * 5) -- the same as 1 + 2 * 5
```

Assignment operators

You can use the = operator to assign a value to a column:

```
UPDATE `myTable` SET `uselessField`=0
```

When you want to assign a value to a variable, you must use the := operator, because the use of = would be ambiguous (is it as assignment or a comparison?)

```
SELECT @myvar := 1
```

You can also use SELECT INTO to assign values to one or more variables.

Comparison operators

Equality

If you want to check if 2 values are equal, you must use the = operator:

```
SELECT True = True -- returns 1
SELECT True = False -- returns 0
```

If you want to check if 2 values are different, you can use the <> or != operators, which have the same meaning:

```
SELECT True <> False -- returns 1
SELECT True != True -- returns 0
```

<> return 1 where = returns 0 and vice versa.

IS and NULL-safe comparison

When you compare a NULL value with a non-NULL value, you'll get NULL. If you want to check if a value is null, you can use IS:

```
SELECT (NULL IS NULL) -- returns 1
SELECT (1 IS NULL) -- returns 0
SELECT (True IS True) -- returns an error!
```

You can check if a value is non-NULL:

```
SELECT (True IS NOT NULL) -- returns 1
```

There is also an equality operator which considers NULL as a normal value, so it returns 1 (not NULL) if both values are NULL and returns 0 (not NULL) if one of the values is NULL:

```
SELECT NULL <=> NULL -- 1
SELECT True <=> True -- 1
SELECT col1 <=> col2 FROM myTable
```

There is not a NULL-safe non-equality operator, but you can type the following:

```
SELECT NOT (col1 <=> col2) FROM myTable
```

IS and Boolean comparisons

IS and IS NOT can also be used for Boolean comparisons. You can use them with the reserved words TRUE, FALSE and UNKNOWN (which is merely a synonym for NULL).

```
SELECT 1 IS TRUE -- returns 1
SELECT 1 IS NOT TRUE -- returns 0
SELECT 1 IS FALSE -- returns 0
SELECT (NULL IS NOT FALSE) -- returns 1: unknown is not false
SELECT (NULL IS UNKOWN) -- returns 1
SELECT (NULL IS NOT UNKNOWN) -- returns 0
```

Greater, Less...

You can check if a value is greater than another value:

```
SELECT 100 > 0 -- returns 1
SELECT 4 > 5 -- return 0
```

You can also check if a value is minor than another value:

```
SELECT 1 < 2 -- returns 1
SELECT 2 < 2 -- returns 0
```

This kind of comparisons also works on TEXT values:

```
SELECT 'a' < 'b' -- returns 1
```

Generally speaking, alphabetical order is used for TEXT comparisons. However, the exact rules are defined by the COLLATION used. A COLLATION defines the sorting rules for a given CHARACTER SET. For example, a COLLATION may be case-sensitive, while another COLLATION may be case-insensitive.

You can check if a value is equal or greater than another value. For example, the following queries have the same meaning:

```
SELECT `a` >= `b` FROM `myTable`
SELECT NOT (`a` < `b`) FROM `myTable`
```

Similarly, you can check if a value is less or equal to another value:

```
SELECT `a` <= `b` FROM `myTable`
```

BETWEEN

If you want to check if a value is included in a given range, you can use the BETWEEN ... AND ... operator. AND doesn't have its usual meaning. Example:

```
SELECT 20 BETWEEN 10 AND 100 -- returns 1
```

The value after BETWEEN and the value after AND are included in the range.

You can also use NOT BETWEEN to check if a value is not included in a range:

```
SELECT 8 NOT BETWEEN 5 AND 10 -- returns 0
```

IN

You can use the IN operator to check if a value is included in a list of values:

```
SELECT 5 IN (5, 6, 7) -- returns 1
SELECT 1 IN (5, 6, 7) -- returns 0
```

You should not include in the list both numbers and strings, or the results may be unpredictable. If you have numbers, you should quote them:

```
SELECT 4 IN ('a', 'z', '5')
```

There is not a theoretical limit to the number of values included in the IN operator.

You can also use NOT IN:

```
SELECT 1 NOT IN (1, 2, 3) -- returns 0
```

Logical operators

MySQL Boolean logic

MySQL doesn't have a real BOOLEAN datatype.

FALSE is a synonym for 0. Empty strings are considered as FALSE in a Boolean context.

TRUE is a synonym for 1. All non-NULL and non-FALSE data are considered as TRUE in a boolean context.

UNKNOWN is a synonym for NULL. The special date 0/0/0 is NULL.

NOT

NOT is the only operator which has only one operand. It returns 0 if the operand is TRUE, returns 1 if the operand is FALSE and returns NULL if the operand is NULL.

```
SELECT NOT 1 -- returns 0
SELECT NOT FALSE -- returns 1
SELECT NOT NULL -- returns NULL
SELECT NOT UNKNOWN -- returns NULL
```

! is a synonym for NOT.

```
SELECT !1
```

AND

AND returns 1 if both the operands are TRUE, else returns 0; if at least one of the operands is NULL, returns NULL.

```
SELECT 1 AND 1 -- returns 1
SELECT 1 AND '' -- return 0
SELECT '' AND NULL -- returns NULL
```

&& is a synonym for AND.

```
SELECT 1 && 1
```

OR

OR returns TRUE if at least one of the operands is TRUE, else returns FALSE; if the two operands are NULL, returns NULL.

```
SELECT TRUE OR FALSE -- returns 1
SELECT 1 OR 1 -- returns 1
SELECT FALSE OR FALSE -- returns 0
SELECT NULL OR TRUE -- returns NULL
```

|| is a synonym for OR.

```
SELECT 1 || 0
```

XOR

XOR (eXclusive OR) returns 1 if only one of the operands is TRUE and the other operand is FALSE; returns 0 if both the operands are TRUE o both the operands are FALSE; returns NULL if one of the operands is NULL.

```
SELECT 1 XOR 0 -- returns 1
SELECT FALSE XOR TRUE -- returns 1
SELECT 1 XOR TRUE -- returns 0
SELECT 0 XOR FALSE -- returns 0
SELECT NULL XOR 1 -- returns NULL
```

Synonyms

- AND can be written as &&
- OR can be written ad ||
- NOT can be written as !

Only NOT (usually) has a different precedence from its synonym. See operator precedence for detail.

Arithmetic operators

MySQL supports operands which perform all basic arithmetic operations.

You can type positive values with a '+', if you want:

```
SELECT +1 -- return 1
```

You can type negative values with a '-'. - is an inversion operand:

```
SELECT -1 -- returns -1
SELECT -+1 -- returns -1
SELECT --1 -- returns 1
```

You can make sums with '+':

```
SELECT 1 + 1 -- returns 2
```

You can make subtractions with '-':

```
SELECT True - 1 -- returns 0
```

You can multiply a number with '*':

```
SELECT 1 * 1 -- returns 1
```

You can make divisions with '/'. Returns a FLOAT number:

```
SELECT 10 / 2 -- returns 5.0000
SELECT 1 / 1 -- returns 1.0000
SELECT 1 / 0 -- returns NULL (not an error)
```

You can make integer divisions with DIV. Resulting number is an INTEGER. No reminder. This has been added in MySQL 4.1.

```
SELECT 10 DIV 3 -- returns 3
```

You can get the reminder of a division with '%' or MOD:

```
SELECT 10 MOD 3 -- returns 1
```

Using + to cast data

You can convert an INTEGER to a FLOAT doing so:

```
SELECT 1 + 0.0 -- returns 1.0
SELECT 1 + 0.000 -- returns 1.000
SELECT TRUE + 0.000 -- returns 1.000
```

You can't convert a string to a FLOAT value by adding 0.0, but you can cast it to an INTEGER:

```
SELECT '1' + 0 -- returns 1
SELECT '1' + FALSE -- returns 1
SELECT <nowiki>''</nowiki> + <nowiki>''</nowiki> -- returns 0
```

Text operators

There are no concatenation operators in MySQL.

Arithmetic operators convert the values into numbers and then perform arithmetic operations, so you can't use + to concatenate strings.

You can use the CONCAT() function instead.

LIKE

The LIKE operator may be used to check if a string matches to a pattern. A simple example:

```
SELECT * FROM articles WHERE title LIKE 'hello world'
```

The pattern matching is usually case insensitive. There are two exceptions:

- when a LIKE comparison is performed against a column which has been declared with the BINARY flag (see CREATE TABLE);
- when the expression contains the BINARY clause:

```
SELECT * 'test' LIKE BINARY 'TEST' -- returns 0
```

You can use two special characters for LIKE comparisons:

- _ means "any character" (but must be 1 char, not 0 or 2)

- **%** means "any sequence of chars" (even 0 chars or 1000 chars)

Note that "\" also escapes quotes (""") and this behaviour can't be changed by the ESCAPE clause. Also, the escape character does not escape itself.

Common uses of LIKE:

- Find titles starting with the word "hello":

```
SELECT * FROM articles WHERE title LIKE 'hello%'
```

- Find titles ending with the word "world":

```
SELECT * FROM articles WHERE title LIKE '%world'
```

- Find titles containing the word "gnu":

```
SELECT * FROM articles WHERE title LIKE '%gnu%'
```

These special chars may be contained in the pattern itself: for example, you could need to search for the "_" character. In that case, you need to "escape" the char:

```
SELECT * FROM articles WHERE title LIKE '\_%' -- titles starting with _
SELECT * FROM articles WHERE title LIKE '\%%' -- titles starting with %
```

Sometimes, you may want to use an escape character different from "\". For example, you could use "/":

```
SELECT * FROM articles WHERE title LIKE '/_%' ESCAPE '/'
```

When you use = operator, trailing spaces are ignored. When you use LIKE, they are taken into account.

```
SELECT 'word' = 'word ' -- returns 1
SELECT 'word' LIKE 'word ' -- returns 0
```

LIKE also works with numbers.

```
SELECT 123 LIKE '%2%' -- returns 1
```

If you want to check if a pattern doesn't match, you can use NOT LIKE:

```
SELECT 'a' NOT LIKE 'b' -- returns 1
```

SOUNDS LIKE

You can use SOUNDS LIKE to check if 2 text values are pronounced in the same way. SOUNDS LIKE uses the SOUNDEX algorithm, which is based on English rules and is very approximate (but simple and thus fast).

```
SELECT `word1` SOUNDS LIKE `word2` FROM `wordList` -- short form
SELECT SOUNDEX(`word1`) = SOUNDEX(`word2`) FROM `wordList` -- long form
```

SOUNDS LIKE is a MySQL-specific extension to SQL. It has been added in MySQL 4.1.

Regular expressions

You can use REGEXP to check if a string matches to a pattern using regular expressions.

```
SELECT 'string' REGEXP 'pattern'
```

You can use RLIKE as a synonym for REGEXP.

Bitwise operators

Bit-NOT:

```
SELECT ~0 -- returns 18446744073709551615
SELECT ~1 -- returns 18446744073709551614
```

Bit-AND:

```
SELECT 1 & 1 -- returns 1
SELECT 1 & 3 -- returns 1
SELECT 2 & 3 -- returns 2
```

Bit-OR:

```
SELECT 1 | 0 -- returns 1
SELECT 3 | 0 -- returns 3
SELECT 4 | 2 -- returns 6
```

Bit-XOR:

```
SELECT 1 ^ 0 -- returns 1
SELECT 1 ^ 1 -- returns 0
SELECT 3 ^ 1 -- returns 2
```

Left shift:

```
SELECT 1 << 2 -- returns 4
```

Right shift:

```
SELECT 1 >> 2 -- 0
```

Language/Import/export

Aside from mysqldump (cf. MySQL/Administration), you can also export / import raw data.

Export data

Data can be exported using the "INTO OUTFILE" keyword

```
SELECT * FROM destinataire INTO OUTFILE '/tmp/test' WHERE id IN (41, 141, 260, 317, 735, 888, 1207, 2211);
```

Beware that the MySQL daemon itself will write the file, not the user you run the MySQL client with. The file will be stored on the server, not on your host. Moreover, the server will need write access to the path you specify (usually, the server can _not_ write in your home directory, e.g.). Hence why we (unsecurely) used `/tmp` in the examples.

You can also use the command line to export data

```
mysql < query.txt > output.txt
```

where query.txt contains an sql-query and the output will be stored in output.txt

Import data

In another database/computer/etc. the data can be imported:

```
LOAD DATA INFILE '/tmp/test' INTO TABLE destinataire;
```

additional options are

```
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
```

to specify how the document is set up and whether there is a header. The columns in the data file can be mapped to the columns of the database table if they do not correspond and it is thus also possible to omit certain columns using a dummy variable:

```
LOAD DATA LOCAL INFILE
'/tmp/test'
INTO TABLE destinataire
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
IGNORE 1 LINES
(
@dummy,
name,
phone_number,
@dummy,
@dummy,
@dummy,
@dummy,
@dummy,
@dummy,
@dummy
)
```

In this example, we only need the second and third column of the data file and store these values in the name and phone_number column of our database table.

Content precisions

To import a .sql which creates a user and its database, one should know if this user already exists on the server, because MySQL doesn't have any `DROP USER IF EXISTS`. On the contrary it works with the databases:

```
DROP DATABASE IF EXISTS `base1`;
CREATE DATABASE `base1` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
USE `base1`;
--DROP USER `user1`@'localhost';
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'p@ssword1';
GRANT USAGE ON *.* TO 'user1'@'localhost' IDENTIFIED BY 'p@ssword1';
GRANT ALL PRIVILEGES ON `user1`.* TO 'user1'@'localhost';
```

PS: if this commande returns *"DROP DATABASE" statements are disabled* with PhpMyAdmin, modify `config.default.php` by switching `$cfg['AllowUserDropDatabase']` to `true`, and empty the navigator cache.

Language/Functions

Syntax

Function names are case insensitive. You can write them as you prefer:

```
SELECT database() -- ok
SELECT DataBase() -- ok
SELECT DATABASE() -- ok
```

If the IGNORE_SPACE SQL_MODE is not set, you can not put a space between the function name and the first parenthesis. It would return a 1064 error. IGNORE_SPACE is usually 0. The reason is that the parser is faster if that flag is disabled. So:

```
SELECT DATABASE () -- usually not accepted
SELECT DATABASE() -- always works fine
```

However, this restriction only applies to the native MySQL functions. UDFs and stored functions may be written with a space after the name.

You can't use a value calculated in the SELECT clause as a constraint in the WHERE clause (its a chicken & egg problem); the WHERE clause is what determines the values in the SELECT clause. What you want is the HAVING clause which is applied *after* all matching rows have been found.

General functions

Type-indipendent functions.

BENCHMARK(times, expression)

Executes expression n times and returns how time it spent. Useful to find bottlenecks in SQL expressions.

```
SELECT BENCHMARK(10000, 'hello');   -- Treatment in 0.0010 sec
```

CAST(value AS type)

Returns value converted in the specified type.

```
SELECT CAST(20130101 AS DATE);  -- 2013-01-01
```

CHARSET(string)

Returns the CHARACTER SET used by string.

```
SELECT CHARSET(20130101);  -- binary
SHOW CHARACTER SET;        -- displays all the different installed CHARACTER SET
```

COALESCE(value, ...)

Returns the first argument which is not NULL. If all arguments are NULL, returns NULL. There must be at least one argument.

```
SELECT COALESCE(NULL, 'hello', NULL);   -- hello
```

COERCIBILITY(string)

Returns the coercibility (between 0 to 5):

```
SELECT COERCIBILITY('hello');   -- 4
```

Coercibility[1]	Meaning	Example
0	Explicit collation	Value with COLLATE clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value
3	System constant	USER() return value
4	Coercible	Literal string
5	Ignorable	NULL or an expression derived from NULL

COLLATION(string)

Returns the COLLATION used by the string.

```
SELECT COLLATION('hello');   -- utf8_general_ci
```

CONNECTION_ID()

Returns the id of the current thread.

```
SELECT CONNECTION_ID();   -- 31
```

CONVERT(value, type)

Returns value converted to the specified type.

```
SELECT CONVERT ('666', UNSIGNED INTEGER)
```

CONVERT(string USING charset)

Converts the passed string to the specified CHARACTER SET.

```
SELECT CONVERT ('This is a text' USING utf8)
```

CURRENT_USER()

Returns the username and the hostname used in the current connection.

```
SELECT CURRENT_USER()
SELECT CURRENT_USER -- it's correct
```

DATABASE()

Returns the current database's name, set with the USE command.

```
SELECT DATABASE()
```

FOUND_ROWS()

After a SELECT with a LIMIT clause and the SQL_CALC_FOUND_ROWS keyword, you can run another SELECT with the FOUND_ROWS() function. It returns the number of rows found by the previous query if it had no LIMIT clause.

```
SELECT SQL_CALC_FOUND_ROWS * FROM stats ORDER BY id LIMIT 10 OFFSET 50
SELECT FOUND_ROWS() AS n
```

GREATEST(value1, value2, ...)

Returns the greatest argument passed.

IF(val1, val2, val3)

If val1 is TRUE, returns val2. If val1 is FALSE or NULL, returns val3.

IFNULL(val1, val2)

If val1 is NULL, returns val2; else, returns val1.

ISNULL(value)

If the value passed is NULL returns 1, else returns 0.

INTERVAL(val1, val2, val3, ...)

Returns the location of the first argument which is greater than the first one, beginning by zero in the integers in parameter:

```
SELECT INTERVAL(10, 20, 9, 8, 7);  -- 0
SELECT INTERVAL(10, 9, 20, 8, 7);  -- 1
SELECT INTERVAL(10, 9, 8, 20, 7);  -- 2
SELECT INTERVAL(10, 9, 8, 7, 20);  -- 3
```

NULLIF(val1, val2)

If val1 = val2, returns NULL; else, returns val1.

LEAST(value1, value2, ...)

Returns the minimum argument passed.

Date and time

There are plenty of date related functions.[2]

```
SELECT * FROM mytable
 WHERE datetimecol >= (CURDATE() - INTERVAL 1 YEAR)  AND
 datetimecol < (CURDATE() - INTERVAL 1 YEAR) INTERVAL 1 DAY;

SELECT IF(DAYOFMONTH(CURDATE()) <= 15,
 DATE_FORMAT(CURDATE(), '%Y-%m-15'),
 DATE_FORMAT(CURDATE() + INTERVAL 1 MONTH, '%Y-%m-15')) AS next15
FROM table;

SELECT YEAR('2002-05-10'), MONTH('2002-05-10'), DAYOFMONTH('2002-05-10')

SELECT PurchaseDate FROM table WHERE YEAR(PurchaseDate) <= YEAR(CURDATE())

SELECT columns FROM table
WHERE start_time >= '2004-06-01 10:00:00' AND end_time <= '2004-06-03 18:00:00'

SELECT * FROM t1
WHERE DATE_FORMAT(datetime_column, '%T') BETWEEN 'HH:MM:SS' AND 'HH:MM:SS'

SELECT Start_time, End_time FROM Table
WHERE Start_time >= NOW() - INTERVAL 4 HOUR

SELECT NOW() + INTERVAL 60 SECOND

SELECT UNIX_TIMESTAMP('2007-05-01'); -- 1177970400
SELECT FROM_UNIXTIME(1177970400); -- 2007-05-01 00:00:00
```

Attention: `convert('17/02/2016 15:49:03',datetime)` or `convert('17-02-2016 15:49:03',datetime)` gives *null*, so an insert request replaces it by the same result as `now()`. This should be `convert('2016-02-17 15:49:03',datetime)` or `convert('2016/02/17 15:49:03',datetime)`.

Aggregate functions

COUNT(field)

If * is given, instead of the name of a field, COUNT() returns the number of rows found by the query. It's commonly used to get the number of rows in a table.

```
SELECT COUNT(*) FROM `antiques`
```

If the DISTINCT keyword is used, identical rows are counted only once.

```
SELECT COUNT(DISTINCT *) FROM `antiques`
```

If a field name is given, returns the number of non-NULL values.

```
SELECT COUNT(`cost`) FROM `antiques`
```

If a field name is given and the DISTINCT keyword is given, returns the number of non-NULL values, and identical values are counted only once.

```
SELECT COUNT(DISTINCT `cost`) FROM `antiques`
```

You can count non-NULL values for an expression:

```
SELECT COUNT(`longitude` + `latitude`) FROM `cities`
```

This returns the number of rows where longitude and latitude are both non-NULL.

MAX(field)

MAX() can be used to get the maximum value for an expression in the rows matching to a query. If no row matches the query, returns NULL.

```
SELECT MAX(`cost`) FROM `antiques`
SELECT MAX(LENGTH(CONCAT(`first_name`, ' ', `last_name`))) FROM `subscribers`
```

MIN(field)

MIN() can be used to get the minimum value for an expression in the rows matching to a query. If no row matches the query, returns NULL.

```
SELECT MIN(`cost`) FROM `antiques`
```

AVG(field)

AVG() can be used to get the average value for an expression in the rows matching to a query. If no row matches the query, returns NULL.

```
SELECT AVG(`cost`) FROM `antiques`
```

SUM(field)

SUM() can be used to get the sum of the values for an expression in the rows matching to a query. If no row matches the query, returns NULL.

If SUM(DISTINCT expression) is used, identical values are added only once. It has been added in MySQL 5.1.

```
SELECT SUM(`cost`) FROM `antiques`
```

GROUP_CONCAT(field)

GROUP_CONCAT() can be used to concatenate values from all records for a group into a single string separated by comma or any additional token you like.

```
CREATE TEMPORARY TABLE p (
      id INTEGER, ptype VARCHAR(10), pname VARCHAR(50)
   );

INSERT INTO p VALUES
    (1,'mp3','iPod'),
    (2,'mp3','Zune'),
    (3,'mp3','ZEN'),
    (4,'notebook','Acer Eee PC'),
    (4,'notebook','Everex CloudBook');

SELECT * FROM p;

SELECT ptype,group_concat(pname)
FROM p
GROUP BY ptype;

SELECT ptype,group_concat(' ',pname)
FROM p
GROUP BY ptype
;
```

Aggregate bit functions

General syntax:

```
FUNCTION_NAME(''expression'')
```

These functions calculate *expression* for each row of the result set and perform the calculation between all the *expression*s. These are bitwise functions. The precision used is 64 bit.

AND

```
SELECT BIT_AND(ip) FROM log
```

OR

```
SELECT BIT_OR(ip) FROM log
```

(returns 0 if there are no rows)

XOR

```
SELECT BIT_XOR(ip) FROM log
```

(returns 0 if there are no rows)

References

1. http://dev.mysql.com/doc/refman/5.0/en/information-functions.html#function_coercibility
2. https://dev.mysql.com/doc/refman/5.5/en/date-and-time-functions.html

Stored Programs

MySQL supports some procedural extensions to SQL. By using them, you can manage the control flow, create loops and use cursors. These features allow you to create stored programs, which may be of 3 kinds:

- Triggers - programs which are *triggered* before / after a certain event involves a table (DELETE, INSERT, UPDATE);
- Events - programs which are executed regularly after some time intervals;
- Stored Procedures - programs which can be called via the CALL SQL command.

MySQL future versions will support stored program written in other languages, not only SQL. You will have the ability to manage new languages as PLUGINs. Also, the stored procedures will be compiled into C code, and thus they will be faster.

Triggers

Managing Triggers

Triggers were added in MySQL 5.0.2. They work on persistent tables, but can't be associated with TEMPORARY tables.

CREATE TRIGGER

To create a new trigger:

```
CREATE TRIGGER `delete_old` AFTER INSERT ON `articles`
    FOR EACH ROW BEGIN
        DELETE FROM `articles` ORDER BY `id` ASC LIMIT 1
    END
```

This example trigger defines a stored program (which is the simple DELETE statement) called `delete_old`. It's automatically fired when a new record is INSERTed into `articles`. It's called after the INSERT, not before. If a single INSERT adds more than one row to the table, `delete_old` is called more than once. The idea is simple: when a new record is created, the oldest record is DELETEd.

A trigger may be executed BEFORE or AFTER a certain SQL statement. This is important because a trigger may execute one or more statements which activate other triggers; so, it may be important to decide their time order, to ensure the database's integrity.

The statement which fires the trigger must be a basic DML command:

- **INSERT**, which includes LOAD DATA and REPLACE
- **DELETE**, which includes REPLACE, but not TRUNCATE
- **UPDATE**

A special case is INSERT ... ON DUPLICATE KEY UPDATE. If the INSERT is executed, both BEFORE INSERT and AFTER INSERT are executed. If the INSERT is not executed, and thus an UPDATE is executed instead, the order of events is the following: BEFORE INSERT, BEFORE UPDATE, AFTER UPDATE.

You can also specify the table's name by using the following syntax:

```
... ON `my_database`.`my_table` ...
```

Triggers' names must be unique in a database. Two tables located in the same database can't be associated to two different triggers with the same name.

Unlike other DBMSs and standard SQL, all triggers are fired FOR EACH ROW, and can't be executed for each statement.

A stored program must be specified between BEGIN and END reserved words. You can't use dynamic SQL here (the PREPARE statement); use can call a stored procedure, instead. If you execute only one statement, you can omit the BEGIN and END words.

You can access to the old value of a field (the value it has before the execution of the statement) and to the new value (the value it has after the execution of the statement. Example:

```
CREATE TRIGGER `use_values` AFTER INSERT ON `example_tab`
    FOR EACH ROW BEGIN
        UPDATE `changelog` SET `old_value`=OLD.`field1`, `new_value`=NEW.`field1` WHERE `backup_tab`.`id`=`example_tab`.`id`
    END
```

DROP TRIGGER

To DROP a trigger you can use the following syntax:

```
DROP TRIGGER `my_trigger`
```

Or:

```
DROP TRIGGER `my_database`.`my_trigger`
```

Or:

```
DROP TRIGGER IF EXISTS `my_trigger`
```

To alter an existing trigger, you must DROP and re-CREATE it.

Metadata

SHOW CREATE TRIGGER

This command returns the CREATE TRIGGER statement used to create the trigger and some information about the settings which may affect the statement.

```
SHOW CREATE TRIGGER delete_old;
```

- **Trigger** - Trigger name
- **sql_mode** - The value of SQL_MODE at the time of the execution of the statement
- **SQL Original Statement**
- **character_set_client**
- **collation_connection**
- **Database Collation**

This statement was added in MySQL 5.1.

SHOW TRIGGERS

If you want to have a list of all the triggers in the current database, you can type the following:

```
SHOW TRIGGERS
```

If you want to have a list of the triggers contained in another database, you can use:

```
SHOW TRIGGERS IN `my_db`
SHOW TRIGGERS FROM `my_db` -- synonym
```

If you want to list the triggers whose name matches to a LIKE expression:

```
SHOW TRIGGERS FROM `my_db` LIKE 'my_%'
```

More complex filters:

```
SHOW TRIGGERS WHERE table='users'
```

You can't use LIKE and WHERE together.

The columns returned by this statement are:

- **Trigger** - Trigger's name
- **Event** - The SQL command that fires the trigger
- **Table** - The table that is associated to the trigger
- **Statement** - The statement that is executed by the trigger
- **Timing** - BEFORE or AFTER
- **Created** - It's always NULL
- **sql_mode** - The SQL_MODE which was set when the trigger was created
- **Definer** - The user who created the trigger
- **character_set_client** - The value of the `character_set_client` variable when the trigger was created
- **collation_connection** - The value of the `collation_connection` variable when the trigger was created
- **Database Collation** - The COLLATION used by the database (and the trigger)

INFORMATION_SCHEMA.TRIGGERS

The INFORMATION_SCHEMA virtual database has a `TRIGGERS` table. It has the following fields:

- **TRIGGER_CATALOG** - What catalog contains the trigger (not implemented yet)
- **TRIGGER_SCHEMA** - What SCHEMA (DATABASE) contains the trigger
- **TRIGGER_NAME** - Trigger's name
- **EVENT_MANIPULATION** - INSERT / UPDATE /DELETE
- **EVENT_OBJECT_CATALOG** - Not implemented yet
- **EVENT_OBJECT_SCHEMA** - SCHEMA containing the table associated to the trigger
- **EVENT_OBJECT_NAME** - Name of the table associated to the trigger

- **ACTION_ORDER** - Not implemented yet
- **ACTION_CONDITION** - Not implemented yet
- **ACTION_STATEMENT** - Statement(s) to be executed when trigger activates
- **ACTION_ORIENTATION** - Not implemented yet
- **ACTION_TIMING** - BEFORE / AFTER
- **ACTION_REFERENCE_OLD_TABLE** - Not implemented
- **ACTION_REFERENCE_NEW_TABLE** - Not implemented
- **ACTION_REFERENCE_OLD_ROW** - Not implemented
- **ACTION_REFERENCE_NEW_ROW** - Not implemented
- **CREATED** - Creation time (not implemented yet)
- **SQL_MODE** - SQL_MODE valid for this trigger's execution
- **DEFINER** - User who created the trigger, in the form 'user@host'
- **CHARACTER_SET_CLIENT** - The value of the `character_set_client` variable when the trigger was created
- **COLLATION_CONNECTION** - The value of the `collation_connection` variable when the trigger was created
- **DATABASE_COLLATION** - The COLLATION used by the database (and the trigger)

Events

Events are also called Scheduled Events or Temporal Triggers. They are planned events which are executed at certain times, or at specified time intervals. They are similar to the UNIX crontab.

Once an Event is started, it must be completely executed. If it is re-activated before it ends its execution, a new instance of the same Event will be created. If this can happen, it may be a good idea to use LOCKs to assure data consistence.

The Event Scheduler is a thread which is permanently in execution. It starts the Events when they must be started. If you don't need Events, you can disable the Event Scheduler. You can do this starting MySQL with the following option:

```
mysqld --event-scheduler=DISABLED
```

Or you can add a line to the my.cnf configuration file:

```
event_scheduler=DISABLED
```

If the Event Scheduler is not disabled, you will be able to turn it ON/OFF runtime. It is controlled by a global system variable:

```
SELECT event_scheduler -- values: ON / OFF / DISABLED
SET GLOBAL event_scheduler = ON
SET GLOBAL event_scheduler = OFF
```

If the Event Scheduler is ON, you can check its status with SHOW PROCESSLIST. It is shown like all other threads. Its `User` is 'event_scheduler'. When it is sleeping, the value for `State` is 'Waiting for next activation'.

Managing Events

You can use the SQL commands CREATE EVENT, ALTER EVENT and DROP EVENT.

CREATE EVENT

The simplest case. We want a SQL command to be executed tomorrow:

```
CREATE EVENT `newevent`
  ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
  DO
    INSERT INTO `mydatabase`.`news` (`title`, `text`) VALUES ('Example!', 'This is not a reale news')
```

The event name must be specified after "EVENT".

If you want to create a task which will be executed only once at a certain time, you need the AT clause. If you don't want to specify an absolute time, but we want the task to be executed when a time interval is passed, "AT CURRENT_TIMESTAMP + INTERVAL ..." is a useful syntax.

If you want to create a recurring task (which will be executed at regular intervals) you need the EVERY clause:

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

You can also specify a start time and/or an end time. The task will be executed at regular intervals from the start time until the end time:

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY INTERVAL 1 DAY
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
      STARTS CURRENT_TIMESTAMP + 1 MONTH
      ENDS CURRENT_TIMESTAMP + 3 MONTH
```

The allowed time units are:

```
YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, YEAR_MONTH, DAY_HOUR, DAY_MINUTE, DAY_SECOND, HOUR_MINUTE, HOUR_SECOND, MINUTE_SECOND
```

The DO clause specify which statement must be executed.

If a task is composed by more than 1 statement, the BEGIN ... END syntax must be used:

```
delimiter |
CREATE EVENT `newevent`
  ON SCHEDULE
    EVERY 1 DAY
  DO
    BEGIN
      DELETE FROM `logs`.`user` WHERE `deletion_time` < CURRENT_TIMESTAMP - 1 YEAR;
      DELETE FROM `logs`.`messages` WHERE `deletion_time` < CURRENT_TIMESTAMP - 1 YEAR;
      UPDATE `logs`.`activity` SET `last_cleanup` = CURRENT_TIMESTAMP;
  END |
delimiter ;
```

If an EVENT with the same name already exists you get an error from the server. To suppress the error, you can use the IF NOT EXISTS clause:

```
CREATE EVENT `newevent2`
  IF NOT EXISTS
  ON SCHEDULE EVERY 2 DAY
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

After the EVENT is expired (when the timestamp specified in the AT clause or in the ENDS clause), MySQL drops the event by default, as it is no more useful. You may want to preserve it from deletion to ALTER it someday and activate it again, or just to have its code somewhere. You may do this with the ON COMPLETION clause:

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  ON COMPLETION PRESERVE
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

Or, you can explicitly tell MySQL to drop it, even if it's not necessary:

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  ON COMPLETION NOT PRESERVE
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

If you don't tell MySQL to preserve the EVENT after it's expired, but it is already expired immediatly after creation (which happens if you specify a past TIMESTAMP in the AT / ENDS clause), the server creates and drop it as you requested. However, in this case it will inform you returning a 1588 warning.

You can also specify if an EVENT must be enabled. This is done by specifying ENABLE, DISABLE or DISABLE ON SLAVES (used to execute the event on the master and not replacate it on the slaves). The EVENT is enabled by default.

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  ON COMPLETION NOT PRESERVE
  DISABLE
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

To modify this behaviour, you will use ALTER EVENT.

You can specify a comment for the EVENT. Comments have a 64 characters limit. The comment must be a literal, not an expression. Example:

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  ON COMPLETION NOT PRESERVE
  DISABLE
  COMMENT 'let\'s optimize some tables!'
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

You can also specify which user must be used to check privileges during the execution of the EVENT. By default, the CURRENT_USER is used. You can specify that explicitly:

```
CREATE DEFINER = CURRENT_USER
  EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

To specify a different user, you must have the SUPER privilege. In that case, you must specify both the username and the host:

```
CREATE DEFINER = 'allen@localhost'
  EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

ALTER EVENT

The ALTER EVENT statement can be used to modify an existing EVENT.

```
CREATE EVENT `newevent2`
  ON SCHEDULE EVERY 2 DAY
  ON COMPLETION NOT PRESERVE
  RENAME TO `example_event`
  DISABLE
  COMMENT 'let\'s optimize some tables!'
  DO
    OPTIMIZE TABLE `mydatabase`.`news`
```

RENAME TO is used to rename the EVENT.

You only need to specify the clauses that you want to change:

```
CREATE EVENT `newevent2` ENABLE;
```

DROP EVENT

You need the EVENT privilege to drop an event. To drop an event you can type:

```
DROP EVENT `event_name`
```

If the EVENT does not exist, you get a 1517 error. To avoid this, you can use the IF EXISTS clause:

```
DROP EVENT IF EXISTS `event_name`
```

If the EVENT needs to be executed only once or just for a known time period, by default MySQL drops it automatically when it is expired (see the ON COMPLETE clause in CREATE EVENT).

Metadata

SHOW CREATE EVENT

This command returns the CREATE EVENT statement used to create the trigger and some information about the settings which may affect the statement.

Syntax:

```
SHOW CREATE EVENT newevent2;
```

- **Event** - Event name.
- **sql_mode** - SQL mode which was in effect when the CREATE EVENT statement was executed.
- **time_zone** - Time zone that was used when the statement was executed.
- **Create Event** - Statement used to create the event.
- **character_set_client**
- **collation_connection**
- **Database Collation**

SHOW EVENTS

The statement shows information about the EVENTs which are in the current database or in the specified database:

```
SHOW EVENTS

SHOW EVENTS FROM `my_nice_db`
SHOW EVENTS IN `my_nice_db` -- synonym

SHOW EVENTS LIKE 'my_%' -- name starts with 'my_'
SHOW EVENTS WHERE definer LIKE 'admin@%' -- filters on any field
```

- **Db** Database name.
- **Name** Event name.
- **Definer** User which created the EVENT and the host he used, in the form user@host.
- **Time zone** Timezone in use for the EVENT. If it never changed, it should be 'SYSTEM', which means: server's timezone.
- **Type** 'ONE TIME' for EVENTs which are executed only once, 'RECURRING' for EVENTs which are executed regularly.
- **Executed At** The TIMESTAMP of the moment the EVENT will be executed. NULL for recursive EVENTs.
- **Interval Value** Number of intervals between EVENT's executions. See next field. NULL for EVENTs which are executed only once.
- **Interval Field** Interval type to wait between EVENTs executions. For example, if `Interval Field` is 'SECOND' and `Interval Value` is 30, the EVENT will be executed every 30 seconds. NULL for EVENTs which are executed only once.
- **Starts** First execution DATETIME for recurring EVENTs. NULL for events which are executed only once.
- **Ends** Last execution DATETIME for recurring EVENTs. NULL for events which are executed only once.
- **Status** ENABLED, DISABLED, or SLAVESIDE_DISABLED. For ENABLED and DISABLED, see above. SLAVESIDE_DISABLED was added in 5.1 and means that the EVENT is enabled on the master but disabled on the slaves.
- **Originator** Id of the server where the EVENT was created. If it has been created on the current server this value is 0. Added in 5.1.
- **character_set_client**
- **collation_connection**
- **Database Collation**

INFORMATION_SCHEMA.EVENTS

The INFORMATION_SCHEMA virtual database has a `EVENTS` table. It's non-standard and has been added in 5.1. EVENTS has the following fields:

- **EVENT_CATALOG** Always NULL (CATALOGs are not implemented in MySQL).
- **EVENT_SCHEMA** Database name.
- **EVENT_NAME** Event name.
- **DEFINER** User which created the EVENT and the host he used, in the form user@host.
- **TIME_ZONE** Timezone in use for the EVENT. If it never changed, it should be 'SYSTEM', which means: server's timezone.
- **EVENT_BODY** Language used to write the routine that will be executed.
- **EVENT_DEFINITION** Routine that will be executed.
- **EVENT_TYPE** 'ONE TIME' for EVENTs which are executed only once, 'RECURRING' for EVENTs which are executed regularly.
- **EXECUTE_AT** The TIMESTAMP of the moment the EVENT will be executed. NULL for recursive EVENTs.
- **INTERVAL_VALUE** Number of intervals between EVENT's executions. See next field. NULL for EVENTs which are executed only once.
- **INTERVAL_FIELD** Interval type to wait between EVENTs executions. For example, if `Interval Field` is 'SECOND' and `Interval Value` is 30, the EVENT will be executed every 30 seconds. NULL for EVENTs which are executed only once.
- **SQL_MODE** SQL mode which was in effect when the EVENT has been created.
- **STARTS** First execution DATETIME for recurring EVENTs. NULL for events which are executed only once.
- **ENDS** Last execution DATETIME for recurring EVENTs. NULL for events which are executed only once.
- **STATUS** ENABLED, DISABLED, or SLAVESIDE_DISABLED. For ENABLED and DISABLED, see above. SLAVESIDE_DISABLED was added in 5.1 and means that the EVENT is enabled on the master but disabled on the slaves.
- **ON_COMPLETION** 'NOT PRESERVE' (the EVENT will be deleted) or 'PRESERVE' (the EVENT won't be deleted'.
- **CREATED** Creation DATETIME.
- **LAST_ALTERED** Last edit's DATETIME. If the EVENT has never been altered, `LAST_ALTERED` has the same value as `CREATED`.
- **LAST_EXECUTED** Last execution TIMESTAMP. If the EVENT has never been executed yet, this value is NULL.
- **EVENT_COMMENT** Comment associated to the EVENT. Is there is no comment, this value is an empty string.
- **ORIGINATOR** Id of the server where the EVENT was created. If it has been created on the current server this value is 0. Added in 5.1.
- **character_set_client**
- **collation_connection**
- **Database Collation**

Stored Routines

Stored Routines are modules written in SQL (with some procedural extensions) which may be called within another statement, using the CALL command.

Stored Routines are called FUNCTIONs if they return a result, or PROCEDUREs if they don't return anything. STORED PROCEDUREs must not be confused with the PROCEDUREs written in C or LUA which can be used in a SELECT statement; STORED FUNCTIONs must not be confused with UDF, even if they both are created with a CREATE FUNCTION statement.

Advantages of Stored Routines

- They reduce network traffic: they may contain many statements, but only one statement need to be sent to invoke them.
- Ability to keep the logic within the database.
- Reusable modules which can be called from external programs, no matter in what language they are written.
- You can modify the Stored Routines without changing your programs.
- The user which invokes a Stored Routine doesn't need to have access to the tables which it reads / writes.
- Calling Stored Routines are faster than executing single statements.

Managing Stored Routines

CREATE PROCEDURE

```
CREATE DEFINER = `root`@`localhost` PROCEDURE `Module1` ( ) NOT DETERMINISTIC NO SQL SQL SECURITY DEFINER OPTIMIZE TABLE wiki1_page;
```

CALL

```
CALL `Module1` ();
```

DROP PROCEDURE

```
DROP PROCEDURE `Module1` ;
```

Modification

```
DROP PROCEDURE `Module1` ;
CREATE DEFINER = `root`@`localhost` PROCEDURE `Module1` ( ) NOT DETERMINISTIC NO SQL SQL SECURITY DEFINER
BEGIN
 OPTIMIZE TABLE wiki1_page;
 OPTIMIZE TABLE wiki1_user;
END
```

Metadata

SHOW FUNCTION / PROCEDURE STATUS

```
SHOW PROCEDURE STATUS;
```

SHOW CREATE FUNCTION / PROCEDURE

```
SHOW CREATE PROCEDURE Module1;
```

INFORMATION_SCHEMA.ROUTINES

The virtual database INFORMATION_SCHEMA has a table called `ROUTINES`, with the functions and procedures information.

INFORMATION_SCHEMA.PARAMETERS

This table contains all the stored functions values.

Procedural extensions to standard SQL

Delimiter

MySQL uses a character as delimiter - MySQL knows that where that character occurs a SQL statement ends and possibly another statement begins. That character is ';' by default. When you create a stored program which contains more than one statements, you enter only one statement: the CREATE command. However, it contains more then one statements in its body, separated with a ';'. In that case, you need to inform MySQL that ';' does not identify the end of the CREATE statement: you need another delimiter.

In the following example, '|' is used as a delimiter:

```
delimiter |
CREATE EVENT myevent
    ON SCHEDULE EVERY 1 DAY
    DO
      BEGIN
        TRUNCATE `my_db`.`my_table`;
        TRUNCATE `my_db`.`another_table`;
      END
delimiter ;
```

Flow control

The keywords are: `IF, CASE, ITERATE, LEAVE LOOP, WHILE, REPEAT`[1].

Loops

WHILE

```
DELIMITER $$
CREATE PROCEDURE counter()
  BEGIN
    DECLARE x INT;
    SET x = 1;
    WHILE x  <= 5 DO
      SET  x = x + 1;
    END WHILE;
    SELECT x;  -- 6
  END$$
DELIMITER ;
```

LOOP

```
DELIMITER $$
CREATE PROCEDURE counter2()
  BEGIN
    DECLARE x INT;
    SET x = 1;
    boucle1: LOOP
      SET  x = x + 1;
        IF x > 5 THEN
          LEAVE boucle1;
      END IF;
    END LOOP boucle1;
    SELECT x;  -- 6
  END$$
DELIMITER ;
```

REPEAT

```
DELIMITER $$
CREATE PROCEDURE counter3()
  BEGIN
    DECLARE x INT;
    SET x = 1;
```

```
    REPEAT
      SET x = x + 1; UNTIL x > 5
    END REPEAT;
    SELECT x;  -- 6
  END$$
DELIMITER ;
```

Cursors

The cursors allow to treat each row differently, but it considerably slows the queries.

```
DELIMITER $$
CREATE PROCEDURE cursor1()
BEGIN
    DECLARE result varchar(100) DEFAULT "";
    DECLARE c1 CURSOR FOR
        SELECT page_title
        FROM wiki1.wiki1_page
        WHERE page_namespace = 0;
    OPEN c1;
    FETCH c1 INTO result;
    CLOSE c1;
    SELECT result;
END;$$
DELIMITER ;
```

They should be declared and open before the loop which should treat every records differently. To know the table end, we should create a handler after the cursor:

```
-- Concatenate all a table column values on a row
DELIMITER $$
CREATE PROCEDURE cursor2()
BEGIN
    DECLARE result varchar(100) DEFAULT "";
    DECLARE total text DEFAULT "";
    DECLARE done BOOLEAN DEFAULT 0;
    DECLARE c2 CURSOR FOR
        SELECT page_title
        FROM wiki1.wiki1_page
        WHERE page_namespace = 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN c2;
    REPEAT
        FETCH c2 INTO result;
        set total = concat(total, result);
    UNTIL done END REPEAT;
    CLOSE c2;
    SELECT total;
END;$$
DELIMITER ;
```

Error handling

A handler declaration permits to specify a treatment in case of error[2]:

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
```

Moreover, the error type can be indicated:

```
DECLARE CONTINUE HANDLER FOR SQLSTATE [VALUE] sqlstate_value
DECLARE CONTINUE HANDLER FOR SQLWARNING
DECLARE CONTINUE HANDLER FOR NOT FOUND
```

References

1. http://dev.mysql.com/doc/refman/5.0/en/flow-control-statements.html
2. http://dev.mysql.com/doc/refman/5.7/en/declare-handler.html

Language/Spatial databases

Principle

When typing the fields, some of them represent graphical objects, and so are considered like as "Spatial". Consequently, they are manipulated by some different request than for the text.

We distinguish eight fields types[1]:

- Geometry
- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString

- MultiPolygon
- GeometryCollection

And six relations between them[2]:

- Contains
- Disjoint
- Equals
- Intersects
- Overlaps
- Within

Requests

References

1. http://dev.mysql.com/doc/refman/5.0/en/spatial-datatypes.html
2. http://dev.mysql.com/doc/refman/5.7/en/spatial-relation-functions-mbr.html

Language/Exercises

Practicing SELECT

Table `list`

ID	Name	Surname	FlatHave	FlatWant
1	Shantanu	Oak	Goregaon	
2	Shantanu	Oak	Andheri	
3	Shantanu	Oak		Dadar
4	Ram	Joshi		Goregaon
5	Shyam	Sharma		Andheri
6	Ram	Naik	Sion	
7	Samir	Shah	Parle	
8	Ram	Joshi	Dadar	
9	Shyam	Sharma	Dadar	

Exercise I - Questions

- Who has a flat in "Goreagon" and who wants to buy one?

This question is ill posed and the listed answer isn't correct. 'and who wants to buy one?' Does this mean wants to buy a flat or wants to buy a flat in Goregaon (which by the way is misspelled in either the question or the table)? The answer is wrong because the question says "AND" and the answer says or. If the question was meant to ask for the names of the people who have a flat in Goregaon AND those who want to buy a flat in Goregaon, then the correct answer to this should be select name, surname from list where flathave="Goregaon" and flatwant="Goregaon"; If the question is meant to ask for names of those who either have OR want a flat in Goregaon, then it would be select name, surname from list where flathave="Goregaon" or flatwant="Goregaon"; If the question is meant to ask for those who have a flat in Goregaon and want to buy a flat, then then answer would be select name, surname from list where flathave="Goregaon" and flatwant<>""; Many of the questions below need revision as well, or the table needs preface information.

- Who has a flat in "Parle" and who wants to buy one?
- Where does "Shantanu Oak" own the flats and where does he want to buy one?
- How many entries have been recorded so far?
- How many flats are there for sale?
- What are the names of our clients?
- How many clients do we have?
- List the customers whose name start with "S"?
- Rearrange the list Alphabetically sorted.

Exercise I - Answers

- `select * from list where FlatHave = "Goregaon" or FlatWant = "Goregaon";`
- `select * from list where FlatHave = "Parle" or FlatWant = "Parle";`
- `select * from list where Name = "Shantanu" and Surname = "Oak";`
- `select count(*) from list;`
- `select count(FlatHave) from list where FlatHave is not null;`
- `select distinct Name, Surname from list;`
- `select count(distinct Name, surname) from list;`
- `select * from list where Name like "S%";`

- `select Surname, Name, FlatHave, FlatWant from list order by Name;`

Table `grades`

ID	Name	Math	Physics	Literature
1	John	68	37	54
2	Jim	96	89	92
3	Bill	65	12	57
4	Jeri	69	25	82

Exercise II - Questions

- A list of all students who scored over 90 on his or her math paper?
- A list of all students who scored more than 85 in all subjects?
- Declare Results: Print the results of all students with result column.
- Find out total marks of all the students.
- What are the average marks of the class for each subject?
- What are the minimum marks in Math?
- What are the maximum marks in Math?
- Who got the highest marks in Math?

Exercise II - Answers

Note: many problems have more than one correct solution.

```
SELECT * FROM grades WHERE math > 90;
SELECT name FROM grades WHERE math > 85 AND physics > 85 AND literature > 85;
SELECT *, IF( (math <= 35 OR physics <= 35 OR literature <= 35), 'fail', 'pass') AS result FROM grades ORDER BY result DESC;
SELECT name, math+physics+literature FROM grades;
SELECT AVG(math), AVG(physics), AVG(literature) FROM grades;
SELECT MIN(math) FROM grades;
SELECT MAX(math) FROM grades;
SELECT * FROM grades ORDER BY math DESC LIMIT 1   -- this is good if we have only one guy with top score.
SELECT * FROM grades where math=max(math);   -- the max() function cannot be used after "where". Such usage results in "ERROR 1111 (HY000): Invalid use
```

These two will work:

```
SELECT name, maths FROM grades WHERE maths = (SELECT MAX(maths) from grades);
SELECT name, maths FROM grades WHERE maths >= ALL (SELECT MAX(maths) from grades);
```

Examples

Finding Duplicates

```
SELECT Vendor, ID, Count(1) as dupes
FROM table_name
GROUP BY Vendor, ID HAVING Count(1) >1

SELECT txt, COUNT(*)
FROM dupes
GROUP BY txt HAVING COUNT(*) > 1;

SELECT id, COUNT( id ) AS cnt,
FROM myTable
GROUP BY id HAVING cnt > 1
```

Remove duplicate entries

Assume the following table and data.

```
CREATE TABLE IF NOT EXISTS dupTest
(pkey int(11) NOT NULL auto_increment,
a int, b int, c int, timeEnter timestamp(14),
PRIMARY KEY  (pkey));

insert into dupTest (a,b,c) values (1,2,3),(1,2,3),(1,5,4),(1,6,4);
```

Note, the first two rows contains duplicates in columns a and b. It contains other duplicates; but, leaves the other duplicates alone.

```
ALTER IGNORE TABLE  dupTest ADD UNIQUE INDEX(a,b);
```

Pivot table

"pivot table" or a "crosstab report"

(Note: this page needs to be wikified)

SQL Characteristic Functions: Do it without "if", "case", or "GROUP_CONCAT". Yes, there is use for this..."if" statements sometimes cause problems when used in combination.

The simple secret, and it's also why they work in almost all databases, is the following functions:

- sign (x) returns -1,0, +1 for values x < 0, x = 0, x > 0 respectively
- abs(sign(x)) returns 0 if x = 0 else, 1 if x > 0 or x < 0
- 1-abs(sign(x)) complement of the above, since this returns 1 only if x = 0

```
  Quick example:   sign(-1) = -1,  abs( sign(-1) ) = 1,  1-abs( sign(-1) ) = 0
```

Data for full example:

```
     CREATE TABLE exams (
       pkey int(11) NOT NULL auto_increment,
       name varchar(15),
       exam int,
       score int,
       PRIMARY KEY  (pkey)
     );

     insert into exams (name,exam,score) values ('Bob',1,75);
     insert into exams (name,exam,score) values ('Bob',2,77);
     insert into exams (name,exam,score) values ('Bob',3,78);
     insert into exams (name,exam,score) values ('Bob',4,80);

     insert into exams (name,exam,score) values ('Sue',1,90);
     insert into exams (name,exam,score) values ('Sue',2,97);
     insert into exams (name,exam,score) values ('Sue',3,98);
     insert into exams (name,exam,score) values ('Sue',4,99);

mysql> select * from exams;
+------+------+------+-------+
| pkey | name | exam | score |
+------+------+------+-------+
|    1 | Bob  |    1 |    75 |
|    2 | Bob  |    2 |    77 |
|    3 | Bob  |    3 |    78 |
|    4 | Bob  |    4 |    80 |
|    5 | Sue  |    1 |    90 |
|    6 | Sue  |    2 |    97 |
|    7 | Sue  |    3 |    98 |
|    8 | Sue  |    4 |    99 |
+------+------+------+-------+
8 rows in set (0.00 sec)

mysql> select name,
sum(score*(1-abs(sign(exam-1)))) as exam1,
sum(score*(1-abs(sign(exam-2)))) as exam2,
sum(score*(1-abs(sign(exam-3)))) as exam3,
sum(score*(1-abs(sign(exam-4)))) as exam4
from exams group by name;

+------+-------+-------+-------+-------+
| name | exam1 | exam2 | exam3 | exam4 |
+------+-------+-------+-------+-------+
| Bob  |    75 |    77 |    78 |    80 |
| Sue  |    90 |    97 |    98 |    99 |
+------+-------+-------+-------+-------+
2 rows in set (0.00 sec)
```

Note, the above pivot table was created with one select statement.

Let's decompose to make the trick clearer, for the second exam:

```
mysql> select name, score, exam, exam-2, sign(exam-2), abs(sign(exam-2)), 1-abs(sign(exam-2)),
       score*(1-abs(sign(exam-2))) as exam2 from exams;
+------+-------+------+--------+--------------+------------------+--------------------+-------+
| name | score | exam | exam-2 | sign(exam-2) | abs(sign(exam-2)) | 1-abs(sign(exam-2)) | exam2 |
+------+-------+------+--------+--------------+------------------+--------------------+-------+
| Bob  |    75 |    1 |     -1 |           -1 |                1 |                  0 |     0 |
| Bob  |    77 |    2 |      0 |            0 |                0 |                  1 |    77 |
| Bob  |    78 |    3 |      1 |            1 |                1 |                  0 |     0 |
| Bob  |    80 |    4 |      2 |            1 |                1 |                  0 |     0 |
| Sue  |    90 |    1 |     -1 |           -1 |                1 |                  0 |     0 |
| Sue  |    97 |    2 |      0 |            0 |                0 |                  1 |    97 |
| Sue  |    98 |    3 |      1 |            1 |                1 |                  0 |     0 |
| Sue  |    99 |    4 |      2 |            1 |                1 |                  0 |     0 |
+------+-------+------+--------+--------------+------------------+--------------------+-------+
8 rows in set (0.00 sec)
```

You may think IF's would be clean but WATCH OUT! Look what the following gives (INCORRECT !!):

```
mysql> select name,
if(exam=1,score,null) as exam1,
if(exam=2,score,null) as exam2,
if(exam=3,score,null) as exam3,
if(exam=4,score,null) as exam4
from exams group by name;

+------+-------+-------+-------+-------+
| name | exam1 | exam2 | exam3 | exam4 |
+------+-------+-------+-------+-------+
| Bob  |    75 |  NULL |  NULL |  NULL |
| Sue  |    90 |  NULL |  NULL |  NULL |
+------+-------+-------+-------+-------+
2 rows in set (0.00 sec)
```

Note: the following does work - is all the maths necessary after all?

```
mysql> SELECT name,
       SUM(IF(exam=1,score,NULL)) AS exam1,
       SUM(IF(exam=2,score,NULL)) AS exam2,
       SUM(IF(exam=3,score,NULL)) AS exam3,
       SUM(IF(exam=4,score,0)) AS exam4
       FROM exams GROUP BY name;
+------+-------+-------+-------+-------+
| name | exam1 | exam2 | exam3 | exam4 |
+------+-------+-------+-------+-------+
| Bob  |    75 |    77 |    78 |    80 |
| Sue  |    90 |    97 |    98 |    99 |
+------+-------+-------+-------+-------+
2 rows in set (0.00 sec)


mysql> select name,
       sum(score*(1-abs(sign(exam-1)))) as exam1,
       sum(score*(1-abs(sign(exam-2)))) as exam2,
       sum(score*(1-abs(sign(exam-3)))) as exam3,
       sum(score*(1-abs(sign(exam-4)))) as exam4,
         sum(score*(1-abs(sign(exam- 2)))) -   sum(score*(1-abs(sign(exam- 1)))) as delta_1_2,
         sum(score*(1-abs(sign(exam- 3)))) -   sum(score*(1-abs(sign(exam- 2)))) as delta_2_3,
         sum(score*(1-abs(sign(exam- 4)))) -   sum(score*(1-abs(sign(exam- 3)))) as delta_3_4
       from exams group by name;
+------+-------+-------+-------+-------+-----------+-----------+-----------+
| name | exam1 | exam2 | exam3 | exam4 | delta_1_2 | delta_2_3 | delta_3_4 |
+------+-------+-------+-------+-------+-----------+-----------+-----------+
| Bob  |    75 |    77 |    78 |    80 |         2 |         1 |         2 |
| Sue  |    90 |    97 |    98 |    99 |         7 |         1 |         1 |
+------+-------+-------+-------+-------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
```

Above delta_1_2 shows the difference between the first and second exams, with the numbers being positive because both Bob and Sue improved their score with each exam. Calculating the deltas here shows it's possible to compare two rows, not columns which is easily done with the standard SQL statements but rows in the original table.

```
mysql>select name,
sum(score*(1-abs(sign(exam-1)))) as exam1,
sum(score*(1-abs(sign(exam-2)))) as exam2,
sum(score*(1-abs(sign(exam-3)))) as exam3,
sum(score*(1-abs(sign(exam-4)))) as exam4,
  sum(score*(1-abs(sign(exam- 2)))) -   sum(score*(1-abs(sign(exam- 1)))) as delta_1_2,
  sum(score*(1-abs(sign(exam- 3)))) -   sum(score*(1-abs(sign(exam- 2)))) as delta_2_3,
  sum(score*(1-abs(sign(exam- 4)))) -   sum(score*(1-abs(sign(exam- 3)))) as delta_3_4,

  sum(score*(1-abs(sign(exam- 2)))) -   sum(score*(1-abs(sign(exam- 1))))  +
  sum(score*(1-abs(sign(exam- 3)))) -   sum(score*(1-abs(sign(exam- 2))))  +
  sum(score*(1-abs(sign(exam- 4)))) -   sum(score*(1-abs(sign(exam- 3))))  as TotalIncPoints
  from exams group by name;

+------+-------+-------+-------+-------+-----------+-----------+-----------+----------------+
| name | exam1 | exam2 | exam3 | exam4 | delta_1_2 | delta_2_3 | delta_3_4 | TotalIncPoints |
+------+-------+-------+-------+-------+-----------+-----------+-----------+----------------+
| Bob  |    75 |    77 |    78 |    80 |         2 |         1 |         2 |              5 |
| Sue  |    90 |    97 |    98 |    99 |         7 |         1 |         1 |              9 |
+------+-------+-------+-------+-------+-----------+-----------+-----------+----------------+
2 rows in set (0.00 sec)
```

TotalIncPoints shows the sum of the deltas.

```
select name,
sum(score*(1-abs(sign(exam-1)))) as exam1,
sum(score*(1-abs(sign(exam-2)))) as exam2,
sum(score*(1-abs(sign(exam-3)))) as exam3,
sum(score*(1-abs(sign(exam-4)))) as exam4,
  sum(score*(1-abs(sign(exam- 2)))) -   sum(score*(1-abs(sign(exam- 1)))) as delta_1_2,
  sum(score*(1-abs(sign(exam- 3)))) -   sum(score*(1-abs(sign(exam- 2)))) as delta_2_3,
  sum(score*(1-abs(sign(exam- 4)))) -   sum(score*(1-abs(sign(exam- 3)))) as delta_3_4,


  sum(score*(1-abs(sign(exam- 2)))) -   sum(score*(1-abs(sign(exam- 1))))  +
  sum(score*(1-abs(sign(exam- 3)))) -   sum(score*(1-abs(sign(exam- 2))))  +
  sum(score*(1-abs(sign(exam- 4)))) -   sum(score*(1-abs(sign(exam- 3))))  as TotalIncPoints,

(sum(score*(1-abs(sign(exam-1)))) +
sum(score*(1-abs(sign(exam-2)))) +
sum(score*(1-abs(sign(exam-3)))) +
sum(score*(1-abs(sign(exam-4)))))/4 as AVG

from exams group by name;

+------+-------+-------+-------+-------+-----------+-----------+-----------+----------------+-------+
| name | exam1 | exam2 | exam3 | exam4 | delta_1_2 | delta_2_3 | delta_3_4 | TotalIncPoints | AVG   |
+------+-------+-------+-------+-------+-----------+-----------+-----------+----------------+-------+
| Bob  |    75 |    77 |    78 |    80 |         2 |         1 |         2 |              5 | 77.50 |
| Sue  |    90 |    97 |    98 |    99 |         7 |         1 |         1 |              9 | 96.00 |
+------+-------+-------+-------+-------+-----------+-----------+-----------+----------------+-------+
2 rows in set (0.00 sec)
```

It's possible to combine Total Increasing Point TotalIncPoints with AVG. In fact, it's possible to combine all of the example cuts of the data into one SQL statement, which provides additional options for displaying data on your page

```
select name,
sum(score*(1-abs(sign(exam-1)))) as exam1,
sum(score*(1-abs(sign(exam-2)))) as exam2,
sum(score*(1-abs(sign(exam-3)))) as exam3,
sum(score*(1-abs(sign(exam-4)))) as exam4,

(sum(score*(1-abs(sign(exam-1)))) +
sum(score*(1-abs(sign(exam-2)))))/2  as AVG1_2,
```

```
(sum(score*(1-abs(sign(exam-2)))) +
sum(score*(1-abs(sign(exam-3)))))/2 as AVG2_3,

(sum(score*(1-abs(sign(exam-3)))) +
sum(score*(1-abs(sign(exam-4)))))/2 as AVG3_4,

(sum(score*(1-abs(sign(exam-1)))) +
sum(score*(1-abs(sign(exam-2)))) +
sum(score*(1-abs(sign(exam-3)))) +
sum(score*(1-abs(sign(exam-4)))))/4 as AVG

from exams group by name;

+------+-------+-------+-------+-------+--------+--------+--------+-------+
| name | exam1 | exam2 | exam3 | exam4 | AVG1_2 | AVG2_3 | AVG3_4 | AVG   |
+------+-------+-------+-------+-------+--------+--------+--------+-------+
| Bob  |    75 |    77 |    78 |    80 |  76.00 |  77.50 |  79.00 | 77.50 |
| Sue  |    90 |    97 |    98 |    99 |  93.50 |  97.50 |  98.50 | 96.00 |
+------+-------+-------+-------+-------+--------+--------+--------+-------+
2 rows in set (0.00 sec)
```

Exam scores are listing along with moving averages...again it's all with one select statement.

Good article on "Cross tabulations" or de-normalizing data to show stats: http://dev.mysql.com/tech-resources/articles/wizard/print_version.html

ADOdb (http://phplens.com/adodb/pivot.tables.html) (PHP) can generate pivot tables using `PivotTableSQL()`.

For Perl, check DBIx-SQLCrosstab (http://search.cpan.org/~gmax/DBIx-SQLCrosstab-1.17/SQLCrosstab.pm).

Table types

Every table is a logical object in a database; but it also needs to physically store its data (records) on the disk and/or in memory. Tables use a Storage Engine to do this. SE are plugins which can be installed or uninstalled into the server (if they're not builtin).

Many operations are requested by the server but physically done by the SE. So, from the SE we choose for a table affects performance, stability, LOCKs type, use of the query cache, disk space required and special features.

In some future versions of MySQL, partitioned tables will be able to use different SE for different partitions.

Let's see which Storage Engine is good for which uses.

Storage Engines

MyISAM and InnoDB

MyISAM does table level locking, while InnoDB does row level locking. In addition to foreign keys, InnoDB offers transaction support, which is absolutely critical when dealing with larger applications. Speed may suffer, particularly for inserts with full transaction guarantees, because all this Foreign Key / Transaction stuff adds overhead.

The default table type for MySQL on Linux is MyISAM, on Windows, normally InnoDB. MyISAM uses table level locking, which means during an UPDATE, nobody can access any other record of the same table. InnoDB however, uses Row level locking. Row level locking ensures that during an UPDATE, nobody can access that particular row, until the locking transaction issues a COMMIT. Many people use MyISAM if they need speed and InnoDB for data integrity.

MyISAM

- **Pros**
  - Fulltext search is currently only available with MyISAM tables
  - Geometric datatypes
  - Sometimes faster reads
  - All numeric key values are stored with the high byte first to allow better index compression
  - Internal handling of one AUTO_INCREMENT column per table is supported. MyISAM automatically updates this column for INSERT and UPDATE operations. This makes AUTO_INCREMENT columns faster (at least 10%)
- **Cons**
  - Table (not row) level locking only
  - No foreign keys constraints (but planned for MySQL 6.x)
  - Slower table checking and restarts after power loss, an issue for those who need high availability

InnoDB

- **Pros**
  - Provides MySQL with a transaction-safe (ACID compliant) storage engine that has commit, rollback, and crash recovery capabilities
  - XA transactions
  - Foreign keys
  - Row level locking
  - Maintains its own buffer pool for caching data and indexes in main memory
  - Faster for some workloads, particularly those where physical ordering by primary key helps or where the automatically built hash indexes speed up record lookups

- Tables can be of any size even on operating systems where file size is limited to 2GB.
- Fast and reliable recovery from power loss.

- **Cons**
  - Data takes more space to store
  - ACID guarantee requires full sync to disk at transaction commit, can be turned off where speed is more important than full ACID guarantees.
  - Data Versioning and transactions add overhead to table management.
  - They can lead to high memory requirements to manage large numbers of locks used in row locking.
  - Indexes are slow to build when they're added after a table has been created. Indexes should therefore be created when data is bulk-loaded.

Overall, InnoDB should be used for with applications that rely highly on data integrity or need transactions, while MyISAM can be used where that is not required or where fulltext indexing is needed. Where speed is more important, both should be tried because which is faster depends on the application.

Drizzle, a MySQL's fork supported by Sun Microsystems, uses InnoDB as its default engine and doesn't support MyISAM.

Merge Table

Synonyms: Merge, MRG_MYISAM

- A MERGE table is a collection of identical MyISAM tables that can be used as one.
- Identical means that all tables have identical column and index information, no deviation of any sort is permitted.

```
CREATE TABLE mumbai (first_name VARCHAR(30), amount INT(10)) TYPE=MyISAM
CREATE TABLE delhi  (first_name VARCHAR(30), amount INT(10)) TYPE=MyISAM
CREATE TABLE total  (first_name VARCHAR(30), amount INT(10)) TYPE=MERGE UNION=(mumbai,delhi)
```

Merges can be used to work around MySQL's or system's filesize limits. In fact those limits affect single MyISAM datafiles, but don't affect the whole Merge table, which doesn't have a datafile.

In the past, in some cases Merge and MyISAM could be used to replace views, which were not supported by MySQL. Merge could be used as a base table and MyISAM tables could be used as views containing part of the base table data. A SELECT on the Merge table returned all the effective data. View support was added in MySQL 5.0, so this use of Merge tables is obsolete.

MEMORY / HEAP

HEAP is the name of this table type before MySQL 4.1. MEMORY is the new, preferred name.

This engine is introduced in version 3.23.

BDB

Synonyms: BDB, BerkleyDB

BDB has been removed from version 5.1 and later due to lack of use.

BerkeleyDB is a family of free software embeddable DBMS's developer by SleepyCat, a company which has been acquired by Oracle. SleepyCat provided a Storage Engine for MySQL called BDB.

BDB supports transactions and page-level locking, but it also has many limitations within MySQL.

BLACKHOLE

Discards all data stored in it but does still write to the binary log, so it is useful in replication scale-out or secure binlog-do filtering situations where slaves aren't trustworthy and for benchmarking the higher layers of the server.

Miscellaneous

For completeness, other storage engines include:

- CSV: simple Comma-Separated Values engine, that uses the CSV format to store data. Used to share database with other CSV-aware applications maybe? Due to the simple nature of its format, indexing is not available.
- EXAMPLE (a stub for developers)
- ISAM (for pre-3.23 backward compatibility, removed in 5.1)

Metadata about Storage Engines

You can get metadata about official MySQL Storage Engines and other Storage Engines which are present on your server, via SQL.

SHOW STORAGE ENGINES

Starting from MySQL 5.0, you can get information about the Storage Engine which you can use using the SHOW STORAGE ENGINES statement.

```
SHOW STORAGE ENGINES
```

The STORAGE word is optional. This command returns a dataset with the following columns:

- **Engine** - Name of the Storage Engine.

- **Support** - Wether the Storage Engine is supported or not. Possible values:
  - 'DEFAULT' - it's supported and it's the default engine;
  - 'YES' - supported;
  - 'DISABLED' - it has been compiled, but MySQL has been started with that engine disabled (possibly with options like --skip-engine-name);
  - 'NO' - not supported.
- **Comment** - Brief description of the engine.
- **Transactions** - Wether the engine supports SQL transactions. Added in MySQL 5.1.
- **XA** - Wether the engine supports XA transactions. Added in MySQL 5.1.
- **Savepoints** - Wether the engine supports savepoints and rollbacks. Added in MySQL 5.1.

INFORMATION_SCHEMA `ENGINES` table

`ENGINES` is a virtual table within the INFORMATION_SCHEMA database. It can be used to get information about Storage Engines. Its columns are the came which are returned by the SHOW ENGINES statement (see above).

ENGINES has been added in MySQL 5.1.5.

HELP statement

If you want more info about an official MySQL Storage Engine, you can use the HELP command:

```
HELP 'myisam'
```

If you are using the command line client, you can omit the quotes:

```
help myisam \g
```

Changing the Storage Engine

SQL

When you want to create a table using a given Storage Engine, you can use the ENGINE clause in the CREATE TABLE command:

```
CREATE TABLE ... ENGINE=InnoDB
```

If the ENGINE clause is not specified, the value of the storage_engine variable will be used. By default it's MyISAM, but you can change it:

```
SET storage_engine=InnoDB
```

Or you can modify the value of default-storage-engine in the my.cnf before starting the MySQL server.

You can also change the Storage Engine of an existing table:

```
ALTER TABLE `stats` ENGINE=MyISAM
```

mysql_convert_table_format

mysql_convert_table_format is a tool provided with MySQL, written in Perl. It converts all the tables contained in the specified database to another Storage Engine.

The syntax is:

```
mysql_convert_table_format [options] database
```

database is the name of the database in which the program will operate. It's mandatory.

Options are:

--help Print a help and exit.

--version Print version number and exit.

--host=host The host on which MySQL is running. Default: localhost.

--port=port TCP port.

--user=user Specify the username.

--password=password Specify the password. As it is insecure (it's visible with the coomand top, for example), you can use an option file, instead.

--type=storage_engine The storage engine that the tables will use after conversion.

--force Don't stop the execution if an error occurs.

--verbose Print detailed information about the conversions.

Example:

```
mysql_convert_table_format --host=localhost --user=root --password=xyz970 --force --type=InnoDB test
```

This command specifies access data (localhost, username, password) and converts all tables within database `test` into InnoDB. If some tables can't be converted, the script skips them and converts the others (--force). *Italic text*

Administration

Installation

Debian packages

The package name is usually *mysql-server*, either directly or as a transitional package for the latest version.

Stable

There are two Debian packages in the current *stable* release:

- mysql-server (http://packages.debian.org/lenny/mysql-server): depends on latest MySQL version
- mysql-server-5.0 (http://packages.debian.org/lenny/mysql-server-5.0): MySQL 5.0

You can install it using this command:

```
apt-get install mysql-server
```

or by installing the package you want using the Synaptic GUI.

Backports

Backports.org may also offers more recent versions.

To install it, you need to add the backports source in your `/etc/apt/sources.list`:

```
deb http://www.backports.org/debian lenny-backports main
```

and then use aptitude:

```
apt-get install -t lenny-backports mysql-server-5.1
```

Uninstall

To simply remove the program:

```
apt-get remove mysql-server
```

To remove the configuration files as well, resulting in a clean environment:

```
apt-get remove --purge mysql-server
```

Debconf will ask you if you want to remove the existing databases as well. Answer wisely!

Fedora Core 5

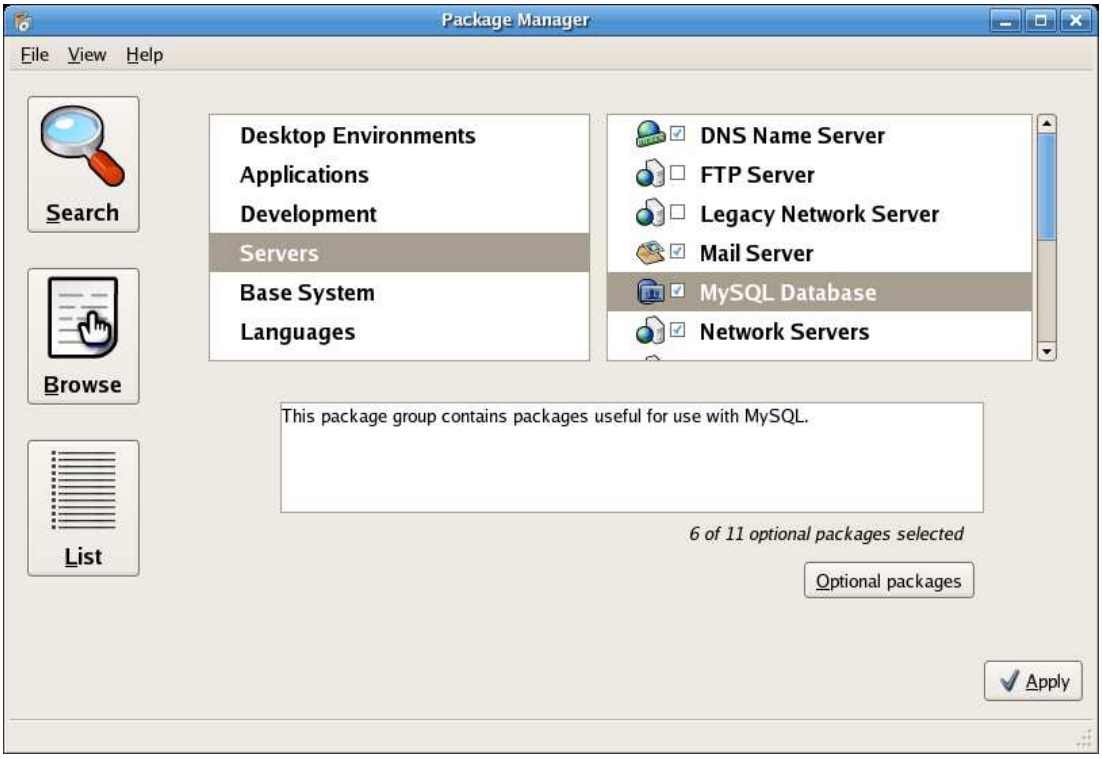
The package name is mysql-server (ftp://ftp.tu-chemnitz.de/pub/linux/fedora-core/5/i386/os/Fedora/RPMS/mysql-server-5.0.18-2.1.i386.rpm).

You can install it using this command:

```
yum install mysql-server
```

which will take care of installing the needed dependencies.

Using *pirut* (Applications->Add/Remove Software), you can also server *MySQL Database* in the *Servers* category:



Gentoo

MySQL is available in the main Portage tree as "dev-db/mysql". You must use the fully qualified ebuild name as "mysql" is made ambiguous by "virtual/mysql"

Command:

```
emerge dev-db/mysql
```

FreeBSD

The stable FreeBSD port is version 5.0 (http://www.freshports.org/databases/mysql50-server/), and beta version 5.1 is also available.

You can install it using this command:

```
cd /usr/ports/databases/mysql50-server/ && make install clean
```

This command will install the MySQL 5.0 server as well as all necessary dependencies (which includes the MySQL client). t

Start the service

Debian

In Debian, you use the `mysql` init script.

```
/etc/init.d/mysql start
/etc/init.d/mysql stop
/etc/init.d/mysql restart
```

If you need to do so in scripts, prefer the `invoke-rc.d` command, which only restart the service if it is launched on system startup. That way, you do not launch a service if it wasn't meant to be run:

```
invoke-rc.d mysql start|stop|restart
```

If you want to control whether to launch MySQL on startup, you can use the `rcconf` package, or update-rc.d:

```
cp /usr/local/mysql/support-files/mysql.server /etc/init.d/anysqlservernamehere
chmod +x /etc/init.d/anysqlservernamehere
update-rc.d anysqlservernamehere defaults
```

Fedora Core

Fedora Core suggests that you use the `service` wrapper, which cleans the environment before to run the service, so that all services run in the same standard environment (for example, the current directory is set to the system root `/`).

```
service mysqld start|stop|restart
```

```
service mysqld --full-restart # means stop, then start - not a direct restart
```

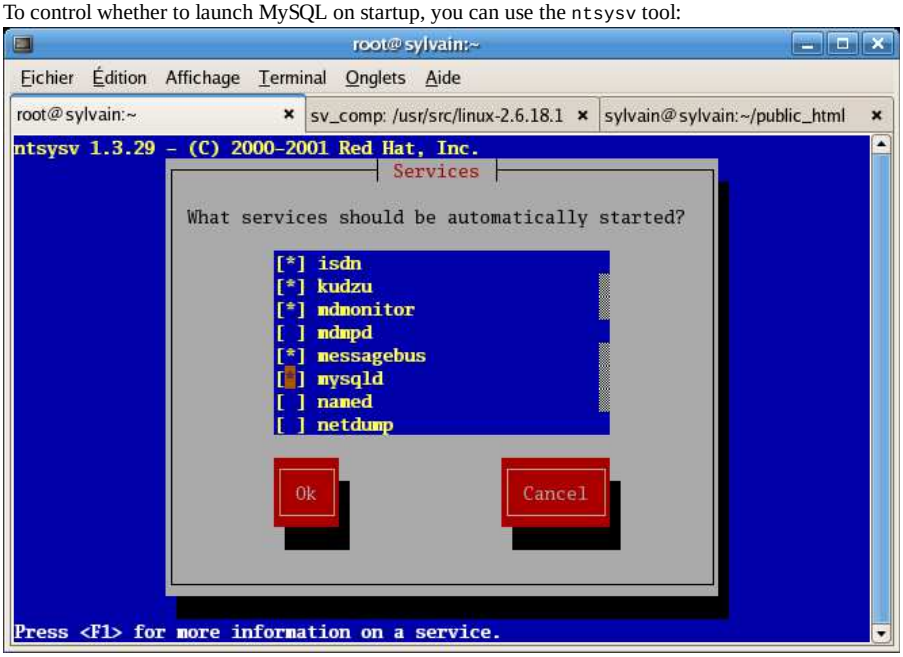
You can also use the `/etc/init.d/mysqld` if needed.

FC5 displays useful hints the first time you launch the MySQL server (i.e. when launching /usr/bin/mysql_install_db):

```
$ service mysqld start
[...]
PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:
/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h localhost password 'new-password'
[...]
```

See the next section about changing passwords.

To control whether to launch MySQL on startup, you can use the `ntsysv` tool:



Client connection

There are two ways to connect to a MySQL server, using Unix sockets and TCP/IP.

The default TCP/IP port is 3306:

```
# grep mysql /etc/services
mysql           3306/tcp                        # MySQL
mysql           3306/udp                        # MySQL
mysql-cluster   1186/tcp                        # MySQL Cluster Manager
mysql-cluster   1186/udp                        # MySQL Cluster Manager
mysql-im        2273/tcp                        # MySQL Instance Manager
mysql-im        2273/udp                        # MySQL Instance Manager
```

As a client, MySQL interprets 'localhost' as 'use the Unix socket'. This means that MySQL won't connect to 127.0.0.1:3306, but will use `/var/run/mysqld/mysqld.sock`:

```
$ mysql -h localhost
mysql> \s
--------------
mysql  Ver 14.12 Distrib 5.0.22, for redhat-linux-gnu (i386) using readline 5.0
[...]
Current user:           sylvain@localhost
[...]
Connection:             Localhost via UNIX socket
[...]
UNIX socket:            /var/lib/mysql/mysql.sock
```

If you really need to connect to MySQL via TCP/IP to the local host without using Unix sockets, then specify '127.0.0.1' instead of 'localhost':

```
$ mysql -h 127.0.0.1
mysql> \s
--------------
mysql  Ver 14.12 Distrib 5.0.22, for redhat-linux-gnu (i386) using readline 5.0
[...]
Current user:           sylvain@localhost
[...]
Connection:             127.0.0.1 via TCP/IP
[...]
TCP port:               3306
```

In both cases, MySQL will understand your machine name as 'localhost' (this is used in the privileges system).

Configuration

Configure /etc/mysql/my.cnf - for heavily loaded databases, for fat databases...; different kinds of connexions (Unix sockets, TCP/IP w/ or w/o SSL, MySQL+SSL licensing issues)

Change the root password

```
$ mysql -u root
mysql> SET PASSWORD = PASSWORD('PassRoot');
```

For more information, see the #SET_PASSWORD section.

Network configuration

```
--bind-address=127.0.0.1 # localhost only
--bind-address=0.0.0.0 # listen on all interfaces
--bind-address=192.168.1.120 # listen on that IP only
```

skip-networking

When you specify `skip-networking` in the configuration, then MySQL will not listen on any port, not even on localhost (127.0.0.1). This means that only programs running on the same machine than the MySQL server will be able to connect to it. This is a common setup on dedicated servers.

The only way to contact MySQL will be to use the local *Unix socket*, such as `/var/run/mysqld/mysqld.sock` (Debian) or `/var/lib/mysql/mysql.sock` (FC5). You can specify where the socket is located using the `socket` parameter in the `[mysqld]` section of the configuration:

```
[mysqld]
...
socket=/var/lib/mysql/mysql.sock
```

Privileges

The MySQL privileges system.

Introduction

MySQL requires you to identify yourself when you connect to the database. You provide the following credentials:

- an identity, composed of:
  - a username
  - a machine name or IP address (detected automatically by the server)
- a password, to prove your identity

Usually, MySQL-aware applications also ask you for a database name, but that's not part of the credentials, because this does not relate to who you are.

MySQL then associates privileges to these credentials; for example, the right to query a given database, add data to another one, create additional databases or remove existing ones, etc.

Who am I?

Once connected, it is not necessarily obvious who MySQL thinks you are. CURRENT_USER() provides this information:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

SHOW GRANTS

Prototype:

```
SHOW GRANTS FOR user
SHOW GRANTS --current user
```

SHOW GRANTS allow you to check the current privileges for a given user. For example, here are the default privileges for user root:

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
1 row in set (0.00 sec)
```

You also use use `SHOW GRANTS;` to check the privileges for the current user.

GRANT

The GRANT command allow you to give (GRANT) privileges to a given user.

DROP USER

```
DROP USER 'mediawiki';
DROP USER 'mediawiki'@'host';
```

Starting with v5.0.2, this removes the associated privileges as well.

With earlier versions you also need to REVOKE its PRIVILEGES manually.

REVOKE

```
REVOKE ALL PRIVILEGES ON database.* FROM 'user'@'host';
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'user'@'host';
```

SET PASSWORD

Prototype:

```
SET PASSWORD [FOR user] = PASSWORD('your_password')
```

If *user* is not specified, the current user is used (this is useful when you connect to mysql using the command line).

Example with an explicit user:

```
SET PASSWORD FOR 'mediawiki'@'localhost' = PASSWORD('ifda8GQg');
```

There is a command-line synonym:

```
mysqladmin password 'your_password'
```

(with the usual connection options `-h` `-u` and `-p`)

However, using passwords on the command line presents a security risk. For example, if root changes his MySQL password:

```
root# mysqladmin password 'K2ekiEk3'
```

Then another user can spy on him by looking at the process list:

```
user$ ps aux | grep mysqladmin
root      7768  0.0  0.1   7044  1516 pts/1    S+   16:57   0:00 mysqladmin password K2ekiEk3
```

Conclusion: don't user `mysqladmin password`.

If you are looking for a way to generate passwords, either secure or easy to remember, try the `pwgen` program (there is a Debian package available):

```
$ pwgen
ooGoo7ba ir4Raeje Ya2veigh zaXeero8 Dae8aiqu rai9ooYi phoTi6gu Yeingo9r
tho9aeDa Ohjoh6ai Aem8chee aheich8A Aelaeph3 eu4Owudo koh6Iema oH6ufuya
[...]
$ pwgen -s # secure
zCRhn8LH EJtzzLRE G4Ezb5BX e7hQ88In TB8hE6nn f8IqdMVQ t7BBDWTH ZZMhZyhR
gbsXdIes hCQMbPE6 XD8Owd0b xitloisw XCWKX9B3 MEATkWHH vW2Y7HnA 3V5ubf6B
[...]
```

Very handy if you manage a lot of accounts :)

MySQL 4.1 password issues

As of version 4.1, MySQL introduced a password-related change.

You'll experience this via errors such as: *Client does not support authentication protocol requested by server; consider upgrading MySQL client.* [1]

If you wish to support older client programs, you need to define the MySQL account password this way:

```
SET PASSWORD [FOR user] = OLD_PASSWORD('your_pass');
```

There is apparently no way to use old passwords with the `GRANT ... IDENTIFIED BY 'password'` syntax.

Alternatively, you can use the `old_passwords` configuration option in your server's `my.cnf`. This means that new passwords will be encoded using the old-style, shorter, less secure format. For example, in Debian Sarge and FC5, the MySQL default configuration enforces old-style password for backward compatibility

with older clients:

```
[mysqld]
...
old_passwords=1
```

1. For example, you can get this error on Debian Sarge's apache+libapache_mod_php4+php4-mysql, the latter depends on libmysqlclient12 aka MySQL 4.0 (`ldd /usr/lib/php4/20020429/mysql.so` gives `libmysqlclient.so.12 => /usr/lib/libmysqlclient.so.12`). If you rely and libmysqlclient14 or later, then your application supports both the old and the new password formats.

Processes

MySQL provides a Unix-like way to show the current server threads and kill them.

SHOW PROCESSLIST

Here is a peaceful MySQL server:

```
mysql> SHOW PROCESSLIST;
+----+-----------+-----------+-----------+---------+------+-------+------------------+
| Id | User      | Host      | db        | Command | Time | State | Info             |
+----+-----------+-----------+-----------+---------+------+-------+------------------+
| 34 | monddprod | localhost | monddprod | Sleep   | 1328 |       | NULL             |
| 43 | root      | localhost | NULL      | Query   |    0 | NULL  | SHOW PROCESSLIST |
+----+-----------+-----------+-----------+---------+------+-------+------------------+
2 rows in set (0.00 sec)
```

`mysqladmin` provides a command-line synonym:

```
$ mysqladmin processlist
+----+-----------+-----------+-----------+---------+------+-------+------------------+
| Id | User      | Host      | db        | Command | Time | State | Info             |
+----+-----------+-----------+-----------+---------+------+-------+------------------+
| 34 | monddprod | localhost | monddprod | Sleep   | 1368 |       |                  |
| 44 | root      | localhost |           | Query   | 0    |       | show processlist |
+----+-----------+-----------+-----------+---------+------+-------+------------------+
```

KILL

If a heavy, nasty query is consuming too much resources on your server, you need to shut it down.

```
TODO: Add a sample SHOW PROCESSLIST output here
```

The brute force way is to restart the server:

```
/etc/init.d/mysql restart
```

A more subtle way is to use SHOW PROCESSLIST to identify the nasty query, and kill it independently of other server threads.

```
mysql> KILL 342;
Query OK, 0 rows affected (0.00 sec)
```

There is also a command-line synonym:

```
$ mysqladmin kill 342
```

Security

Basic security: firewall (iptables), SELinux? also some words about: do not store passwords as cleartext

Backup

Backup/recovery and import/export techniques.

mysqldump

```
mysqldump --opt -h 192.168.2.105 -u john -p'****' mybase | gzip > mybase-`date +%Y%m%d`.sql.gz
```

This creates the `mybase-20061027.sql.gz` file.

`--opt` is the magical option that uses all the options that are generally useful. In recent versions of mysqldump, it is even enabled by default, so you need not type it. `--opt` means `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset` - so it will lock tables during the backup for consistency, add DROP TABLE statements so the dump can be applied without cleaning the target database, will use the most efficient ways to perform the INSERTs and specify the charset (latin1, Unicode/UTF-8...) used.

If you don't provide a database to mysqldump, you'll get a backup containing all databases - which is less easy to use for restoring a single database later on.

Daily rotated mysqldump with logrotate

We're using logrotate in a slightly non-standard way to keep a batch of dumps. Each day, logrotate will cycle the dumps so as to keep the last N dumps, removing old backups automatically, and generating the new one immediately through a postrotate hook.

The following configuration keeps 2 months of daily backups:

```
/dumps/mybase.sql.gz {
        rotate 60
        dateext
        daily
        nocompress
        nocopytruncate
        postrotate
          HOME=/root mysqldump --opt mybase | gzip > /dumps/mybase.sql.gz
        endscript
}
```

Cf. logrotate(8) in the GNU/Linux man pages for more information.

Variant to backup all databases at once:

```
/dumps/*/*.sql.gz {
        daily
        rotate 20
        dateext
        nocompress
        sharedscripts
        create
        postrotate
                export HOME=/root
                for i in $(mysql --batch --skip-column-names -e 'SHOW DATABASES' | grep -vE '^information_schema|performance_schema$'); do
                        if [ ! -e /dumps/$i ]; then mkdir -m 700 /dumps/$i; fi
                        mysqldump --events $i | gzip -c > /dumps/$i/$i.sql.gz
                done
        endscript
}
```

Setup:

- Create your `~/.my.cnf` for password-less database access
- Place the logrotate configuration file above in the `/etc/logrotate.d/` directory
- Bootstrap the first dump:
  - `mkdir -m 700 /dumps`
  - `mkdir -m 700 /dumps/mybase`
  - `touch /dumps/mybase/mybase.sql.gz`
  - `logrotate -f /etc/logrotate.d/mysql-dumps`
- Check the dump using `zcat /dumps/mybase.sql.gz`.

Comments on the code: `HOME=/root` is needed for systems (such as FC5) that set `HOME=/` in their cron, which prevents mysqldump from finding the `.my.cnf` configuration. We also use `| gzip` instead of logrotate's `compress` option for disk I/O efficiency (single-step).

In production, you'll get something like this:

```
# ls -lt /dumps
total 16520
-rw-r----- 1 root clisscom 2819533 mar  2 06:25 clisscom.sql.gz
-rw-r----- 1 root clisscom 2815193 mar  1 06:25 clisscom.sql.gz-20100302
-rw-r----- 1 root clisscom 2813579 fév 28 06:26 clisscom.sql.gz-20100301
-rw-r----- 1 root clisscom 2812251 fév 27 06:25 clisscom.sql.gz-20100228
-rw-r----- 1 root clisscom 2810803 fév 26 06:25 clisscom.sql.gz-20100227
-rw-r----- 1 root clisscom 2808785 fév 25 06:25 clisscom.sql.gz-20100226
...
```

Beware that the date in the filename is the date of the rotation, not the date of the dump. Using `dateext` helps with remote backups, because filenames don't change daily, not you avoid re-downloading all of `/dumps` each time.

Remote mysqldump using CGI

mysqldump can be found sometimes in shared-hosting facilities. You can use a simple CGI script to get a direct dump:

```
#!/bin/sh

echo "Content-Type: application/x-tar"
echo "Content-Encoding: x-gzip"
echo ""

mysqldump --host=mysql.hosting.com --user=john --password=XXXXX my_base | gzip 2>&1
```

You can then get it with your browser or wget:

```
$ wget -O- --quiet http://localhost/~sylvain/test2.cgi > base-`date +%Y%m%d`.sql.gz
```

You can even re-inject it on-the-fly in your local test database:

```
$ wget -O- --quiet http://localhost/~sylvain/test2.cgi | gunzip | mysql test_install -u myself -pXXXX
```

Protect the script with a `.htaccess`, write a `.netrc` for wget to use, and you'll have a simple, unattended way to grap a backup even without command-line access. This allows to gain time when grabing a dump (compared to using phpMyAdmin) and to setup remote automated backups (no interaction is needed).

Something similar should be feasible in PHP provided you have access to exec().

Exporting a single table

If you need to import/export a table, not a complete database, check MySQL/Language#Import_.2F_export.

Binary logs

Binary logs are a mechanism to keep track of everything that happens on the MySQL server (forensics), allowing to replay the same sequence of commands on a different computer (master/slave replication), or at a later time (crash recovery).

On Debian they are stored in `/var/log/mysql/mysql-bin.0*`.

To view the SQL commands in a binary log, you use the `mysqlbinlog` command:

```
mysqlbinlog /var/log/mysql/mysql-bin.000001
```

For the crash recovery to be useful, binary logs are usually stored on a different computer (via a NFS mount, for example). Note that it is meant to recover the *full* mysql server, not just one database. You could attempt to filter the log by database, but this isn't straightforward.

So in order use binary logs as a recovery plan, you usually combine them with a full standard backup:

```
mysqldump -A | gzip > all.sql.gz
```

To flush/reset the logs at the same time (TODO: test):

```
mysqldump -A --master-data --flush-logs | gzip > all.sql.gz
```

To recover you'll just combine the two sources (preferably, disable binary logging in the server configuration during the recovery, and re-enable it right after.):

```
(zcat all.sql.gz && mysqlbinlog /var/log/mysql/mysql-bin.0*) | mysql
```

Logs

Where interesting logs are located, common errors to look at. For example:

```
tail -f /var/log/mysql.log
```

Admin Tools

Various third-party graphical interfaces and utilities.

Web interfaces

- phpMyAdmin (wikipedia: phpMyAdmin)
- eSKUeL (http://eskuel.sourceforge.net/): an alternative to phpMyAdmin
- MySQL on Servers Support (http://www.runmapglobal.com/blog/mysql-databases-on-dedicated-servers/)

Desktop GUI

- MySQL Administrator: (http://mysql.com/products/tools/administrator/) from MySQL AB. If you want to create real backups, though, do not use this, since it runs backups using `at` on the client machine - which is likely not to be online every day.

Replication

What is replication

Replication means that data written on a master MySQL will be sent to separate server and executed there.

Applications:

- backups

- spread read access on multiple servers for scalability
- failover/HA

Replication types:

- Asynchronous replication (basic master/slave)
- Semi-asynchronous replication (asynchronous replication + enforce 1 slave replication before completing queries)

Replication configurations:

- standard: master->slave
- dual master: master<->master

In Master-Master replication both hosts are masters and slaves at the same time. ServerA replicates to serverB which replicates to serevrA. There are no consistency checks and even with auto_increment_increment/auto_increment_offset configured both servers should not be used for concurrent writes.

Asynchronous replication

That's the most simple replication. A master writes a binary log file, and slaves can read this log file (possibly selectively) to replay the query statements. It's asynchronous, which mean the master and slaves may have different states at a specific point of time; also this setup can survive a network disconnection.

Configuration on the master

In `/etc/mysql/my.cnf`, in the `[mysqld]` section:

- Define a server identifier (detects loops?); customarily we'll use `1` for the server, but it can be different:

```
server-id = 1
```

- Replication is based on binary logs, so enable them:

```
log-bin
# or log-bin = /var/log/mysql/mysql-bin.log
```

Create a new user for the slave to connect with:

```
CREATE USER 'myreplication';
SET PASSWORD FOR 'myreplication' = PASSWORD('mypass');
GRANT REPLICATION SLAVE ON *.* to 'myreplication';
```

Verify your server identifier:

```
SHOW VARIABLES LIKE 'server_id';
```

Configuration on each slave

In `/etc/mysql/my.cnf`, in the `[mysqld]` section:

- Define a server identifier, different than the master (and different than the other slaves):

```
server-id = 2
```

- Verify with:

```
SHOW VARIABLES LIKE 'server_id';
```

- You can also declare the slave hostname to the master (cf. `SHOW SLAVE HOSTS` below):

```
report-host=slave1
```

Declare the master:

```
CHANGE MASTER TO MASTER_HOST='master_addr', MASTER_USER='myreplication', MASTER_PASSWORD='mypass';
```

If setting up replication from backup, specify start point (add to previous command):

```
MASTER_LOG_FILE='<binary_log_from_master>', MASTER_LOG_POS=<master_binary_log_position>;
```

Start the replication:

```
START SLAVE;
```

This will create a file named `master.info` in your data directory, typically `/var/lib/mysql/master.info`; this file will contain the slave configuration and

status.

TODO:

```
Oct 15 21:11:19 builder mysqld[4266]: 101015 21:11:19 [Warning] Neither --relay-log nor --relay-log-index were used; so
  replication may break when this MySQL server acts as a slave and has his hostname changed!! Please use
  '--relay-log=mysqld-relay-bin' to avoid this problem.
```

Check the replication

On the slave

On a slave, type:

```
SHOW SLAVE STATUS;
```

Or more for a more readable (line-based) output:

```
SHOW SLAVE STATUS\G
```

Example:

```
*************************** 1. row ***************************
            Slave_IO_State:
               Master_Host: master_addr
               Master_User: myreplication
               Master_Port: 3306
...
```

Check in particular:

```
          Slave_IO_Running: Yes
         Slave_SQL_Running: Yes
```

You can suspect the asynchronous nature of the replication:

```
     Seconds_Behind_Master: 0
```

See also:

```
mysql> SHOW GLOBAL VARIABLES LIKE "%SLAVE%";
```

On the master

You can see a connection from the slave in the process list.

```
mysql> SHOW PROCESSLIST\G
[...]
*************************** 6. row ***************************
     Id: 14485
   User: myreplication
   Host: 10.1.0.106:33744
     db: NULL
Command: Binlog Dump
   Time: 31272
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

If you enabled `report-host`, the slave is also visible in:

```
mysql> SHOW SLAVE HOSTS;
+-----------+---------+------+------------------+-----------+
| Server_id | Host    | Port | Rpl_recovery_rank | Master_id |
+-----------+---------+------+------------------+-----------+
|         2 | myslave | 3306 |                 0 |         1 |
+-----------+---------+------+------------------+-----------+
1 row in set (0.00 sec)
```

Consistency

Note that this replication is a simple replay, similar to feeding a `mysqldump` output to the `mysql` client. Consequently, to maintain the consistency:

- Avoid writing critical data on the slave
- Start the replication with identical initial data on both the master and the slave
- To test: we suspect it would be best to use the same version of MySQL on the master and slaves

Fixing

By default, replicate will stop if it meets an error. This can happen if your master and slaves were not consistent in the beginning, or due to a network error causing a malformed query.

In this case, you'll get a trace in the system log (typically `/var/log/syslog`):

```
Oct 15 21:11:19 builder mysqld[4266]: 101015 21:11:19 [ERROR] Slave: Error 'Table 'mybase.form'
  doesn't exist' on query. Default database: 'mybase'.  Query:
  'INSERT INTO `form` (`form_id`,`timestamp`,`user_id`) VALUES ('abed',1287172429,0)',
  Error_code: 1146
```

The best way is to reset the replication entirely.

You can also fix the mistake manually, and then ask MySQL to skip `1` statement this way:

```
STOP SLAVE;
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1;
START SLAVE;
```

You can set `SQL_SLAVE_SKIP_COUNTER` to any number, e.g. `100`. Beware that in this case, it will skip both valid and invalid statements, not only errors.

Another way to fix broken replication is to use Maatkit tools.

- mk-slave-restart (to restart replication on slave if there are more errors and `SQL_SLAVE_SKIP_COUNTER` can't help)
- mk-table-checksum (to perform checksumming of tables on master and slave)
- mk-table-sync (to sync slave with master based on stats generated by mk-table-checksum)

Uninstalling

To erase the replication:

- Type:

```
mysql> RESET SLAVE;
```

- Note: at this point, MySQL paused the slave and replaced the configuration with default values. The `master.info` file was also removed.
- Restart MySQL to clear all configuration.

Warning: `STOP SLAVE` will stop replication. It can be started manually again or (by default) it will automatically resume if you restart the MySQL server. To avoid auto start of replication during process of startup, add to your configuration file:

```
slave-skip-start
```

If you want to stop the replication for good (and use the server for another purpose), you need to reset the configuration as explained above.

At this point your slave configuration should be completely empty:

```
mysql> SHOW SLAVE STATUS;
Empty set (0.00 sec)
```

Optimization

Before Starting To Optimise

When the database seems to be "slow" first consider all of the following points as e.g. making a certain query absolutely unnecessary by simply using a more sophisticated algorithm in the application is always the most elegant way of optimising it :)

1. Finding the bottleneck (CPU, memory, I/O, which queries)
2. Optimising the application (remove unnecessary queries or cache PHP generated web pages)
3. Optimising the queries (using indices, temporary tables or different ways of joining)
4. Optimising the database server (cache sizes etc)
5. Optimising the system (different filesystem types, swap space and kernel versions)
6. Optimising the hardware (sometimes indeed the cheapest and fastest way)

To find those bottlenecks the following tools have been found to be helpful:

vmstat
: to quickly monitor cpu, memory and I/O usage and decide which is the bottleneck

top
: to check the current memory and cpu usage of mysqld as well as of the applications

mytop
: to figure out which queries cause trouble

mysql-admin (the GUI application, not to confuse with mysqladmin)
: to monitor and tune mysql in a very convenient way

mysqlreport (http://hackmysql.com/mysqlreport)

which output should be use as kind of step by step check list

Using these tools most applications can also be categorised very broadly using the following groups:

- I/O based and reading (blogs, news)
- I/O based and writing (web access tracker, accounting data collection)
- CPU based (complex content management systems, business apps)

Optimising the Tables

Use the following command regularly to reorganize the disk space which reduces the table size without deleting any record[1]:

```
OPTIMIZE TABLE MyTable1
```

Moreover, when creating the tables, their smallest types are preferable. For example:

- if a number is always positive, choose an `unsigned` type to be able to store twice more into the same amount of bytes.
- to store the contemporaneous dates (from 1970 to 2038), it's better to take a `timestamp` on four bytes, than a `datetime` on 8.[2]

Optimising the Queries

Comparing functions with BENCHMARK

The BENCHMARK() function can be used to compare the speed of MySQL functions or operators. For example:

```
mysql> SELECT BENCHMARK(100000000, CONCAT('a','b'));
+---------------------------------------+
| BENCHMARK(100000000, CONCAT('a','b')) |
+---------------------------------------+
|                                     0 |
+---------------------------------------+
1 row in set (21.30 sec)
```

However, this cannot be used to compare queries:

```
mysql> SELECT BENCHMARK(100, SELECT `id` FROM `lines`);
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server version for
the right syntax to use near 'SELECT `id` FROM `lines`)' at line 1
```

As MySQL needs a fraction of a second just to parse the query and the system is probably busy doing other things, too, benchmarks with runtimes of less than 5-10s can be considered as totally meaningless and equally runtimes differences in that order of magnitude as pure chance.

Analysing functions with EXPLAIN

When you precede a SELECT statement with the keyword EXPLAIN, MySQL explains how it would process the SELECT, providing information about how tables are joined and in which order. This allows to place some eventual hints in function.

Using and understanding EXPLAIN is essential when aiming for good performance therefore the relevant chapters of the official documentation are a mandatory reading!

A simple example

The join of two table that both do not have indices:

```
mysql> explain SELECT * FROM a left join b using (i) WHERE a.i < 2;
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | a     | ALL  | NULL          | NULL | NULL    | NULL |    4 | Using where |
|  1 | SIMPLE      | b     | ALL  | NULL          | NULL | NULL    | NULL |    3 |             |
+----+-------------+-------+------+---------------+------+---------+------+------+-------------+
2 rows in set (0.01 sec)
```

Now the second table gets an index and the explain shows that MySQL now knows that only 2 of the 3 rows have to be used.

```
mysql> ALTER TABLE b ADD KEY(i);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> explain SELECT * FROM a left join b using (i) WHERE a.i < 2;
+----+-------------+-------+------+---------------+------+---------+----------+------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref      | rows | Extra       |
+----+-------------+-------+------+---------------+------+---------+----------+------+-------------+
|  1 | SIMPLE      | a     | ALL  | NULL          | NULL | NULL    | NULL     |    4 | Using where |
|  1 | SIMPLE      | b     | ref  | i             | i    | 5       | test.a.i |    2 |             |
+----+-------------+-------+------+---------------+------+---------+----------+------+-------------+
2 rows in set (0.00 sec)
```

Now the first table also gets an index so that the WHERE condition can be improved and MySQL knows that only 1 row from the first table is relevant before even trying to search it in the data file.

```
mysql> ALTER TABLE a ADD KEY(i);
```

```
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> explain SELECT * FROM a left join b using (i) WHERE a.i < 2;
+----+-------------+-------+-------+---------------+------+---------+----------+------+-------------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref      | rows | Extra       |
+----+-------------+-------+-------+---------------+------+---------+----------+------+-------------+
|  1 | SIMPLE      | a     | range | i             | i    | 5       | NULL     |    1 | Using where |
|  1 | SIMPLE      | b     | ref   | i             | i    | 5       | test.a.i |    2 |             |
+----+-------------+-------+-------+---------------+------+---------+----------+------+-------------+
2 rows in set (0.02 sec)
```

Optimising The MySQL Server

Status and server variables

MySQL can be monitored and tuned by watching the **status-variables** and setting the **server-variables** which can both be global or per session. The status-variables can be monitored by *SHOW [GLOBAL|SESSION] STATUS [LIKE '%foo%']* or *mysqladmin [extended-]status*. The server-variables can be set in the */etc/mysql/my.cnf* file or via *SET [GLOBAL|SESSION] VARIABLE foo := bar* and be shown with *mysqladmin variables* or *SHOW [GLOBAL|SESSION] VARIABLES [LIKE '%foo%']*.

Generally status variables start with a capital letter and server variables with a lowercase one.

When dealing with the above mentioned per-session system variables it should always be considered that those have to be multiplied by *max_connections* to estimate the maximal memory consumption. Failing to do so can easily lead to server crashes at times of load peaks when more than usual clients connect to the server! A quick and dirty estimation can be made with the following formular:

```
  min_memory_needed = global_buffers + (thread_buffers * max_connections)
```

```
  global_buffers:
      key_buffer
      innodb_buffer_pool
      innodb_log_buffer
      innodb_additional_mem_pool
      net_buffer
```

```
  thread_buffers:
      sort_buffer
      myisam_sort_buffer
      read_buffer
      join_buffer
      read_rnd_buffer
```

Note: Especially when dealing with server settings, all information should be verified in the respective chapters of the official documentation as these are subject of change and the authors of this text lack confirmed knowledge about how the server works internally.

Index / Indices

Indices are a way to locate elements faster. This works for single elements as well as range of elements.

Experiment

Note: when you make your time tests, make sure the query cache is disabled (`query_cache_type=0` in my.cnf) to force recomputing your queries each time you type them instead of just taking the pre-computed results from the cache.

Let's run the following Perl program:

```
#!/usr/bin/perl

use strict;

print "DROP TABLE IF EXISTS weightin;\n";
print "CREATE TABLE weightin (
    id INT PRIMARY KEY auto_increment,
    line TINYINT,
    date DATETIME,
    weight FLOAT(8,3)
);\n";


# 2 millions records, interval = 100s
for (my $timestamp = 1000000000; $timestamp < 1200000000; $timestamp += 100) {
    my $date = int($timestamp + rand(1000) - 500);
    my $weight = rand(1000);
    my $line = int(rand(3)) + 1;
    print "INSERT INTO weightin (date, line, weight) VALUES (FROM_UNIXTIME($date), $line, $weight);\n";
}
```

What does it do? It simulate the data feeds from an industrial lines that weight stuff at regular intervals so we can compute the average material usage. Over time lots of records are piling up.

How to use it?

```
mysql> CREATE DATABASE industrial
$ perl generate_huge_db.pl | mysql industrial
real   6m21.042s
```

```
user    0m37.282s
sys     0m51.467s
```

We can check the number of elements with:

```
mysql> SELECT COUNT(*) FROM weightin;
+----------+
| count(*) |
+----------+
|  2000000 |
+----------+
1 row in set (0.00 sec)
```

The size must be important:

```
$ perl generate_huge_db.pl > import.sql
$ ls -lh import.sql
-rw-r--r-- 1 root root 189M jun 15 22:08 import.sql

$ ls -lh /var/lib/mysql/industrial/weightin.MYD
-rw-rw---- 1 mysql mysql 35M jun 15 22:17 /var/lib/mysql/industrial/weightin.MYD

$ time mysqldump industrial > dump.sql
real    0m9.599s
user    0m3.792s
sys     0m0.616s
$ ls -lh dump.sql
-rw-r--r-- 1 root root 79M jun 15 22:18 dump.sql

$ time mysqldump industrial | gzip > dump.sql.gz
real    0m17.339s
user    0m11.897s
sys     0m0.488s
$ ls -lh dump.sql.gz
-rw-r--r-- 1 root root 22M jun 15 22:19 dump.sql.gz
```

Incidentally restoring from the dump is way faster, because it uses extended inserts!

```
# time zcat dump.sql.gz | mysql industrial
real    0m31.772s
user    0m3.436s
sys     0m0.580s
```

This SQL command will scan all records to get a total sum:

```
mysql> SELECT SUM(*) FROM weightin;
```

Let's say we need to compute the total material used during January 1st 2008:

```
mysql> SELECT COUNT(*), SUM(poids) FROM pesee WHERE date >= '2008-01-01' AND date < '2008-01-02';
```

MySQL will also need to browse the entire database, even for this tiny amount of records. This is because records can be anywhere: at the bottom, at the end, in the middle, nothing guarantees that the records are ordered.

To improve this, we can add an index to the 'date' field. This means MySQL will create a new hidden table with all the date sorted chronologically, and store their offset (position) in the 'weightin' table to retrieve the full record.

Because the index is sorted, it's way faster for MySQL to locate a single record (using a binary search algorithm) or even a range of data (find the first and last element, the range is in-between).

To add the index:

```
ALTER TABLE weightin ADD INDEX (date);
```

The index doesn't work if the query needs computer on the field (e.g. `TIME(date)`) but works for ranges (e.g. `WHERE date < '2008-01-02'`).

You can notice that the .MYD file grew:

```
$ ls -lh /var/lib/mysql/industrial/
-rw-rw---- 1 mysql mysql  49M jun 15 22:36 weightin.MYI
```

That's were MySQL stores the indices. Initially there was an index for the 'id' field, which the case for all primary keys.

Another example

Another example: let's say we want to optimise this query:

```
mysql> SELECT DISTINCT line FROM weightin;
```

We can do so by adding an index on the 'line' field, in order to group the doubles together, which will avoid the query to rescan the whole table to localize them:

```
ALTER TABLE weightin ADD INDEX (line);
```

The index file grew:

```
-rw-rw---- 1 mysql mysql  65M jun 15 22:38 weightin.MYI
```

General considerations

The first and foremost question that is always asked for SELECT queries is always if indices (aka "keys") are configured and if they are, whether or not they are actually be used by the database server.

1. Check if the indices are actually used

Individual queries can be checked with the "EXPLAIN" command. For the whole server the "Sort_%" variables should be monitored as they indicate how often MySQL had to browse through the whole data file because there was no usable index available.

2. Are the indices buffered

Keeping the indices in memory improves read performance a lot. The quotient of "Key_reads / Key_read_requests" tells how often MySQL actually accessed the index file on disk when it needed a key. Same goes for Key_writes, use mysqlreport to do the math for you here. If the percentage is too high, key_buffer_size for MyISAM and innodb_buffer_pool_size for InnoDB are the corresponding variables to tune.

The Key_blocks_% variables can be used to see how much of the configured key buffer is actually used. The unit is 1KB if not set otherwise in key_cache_block_size. As MySQL uses some blocks internally, key_blocks_unused has to be checked. To estimate how big the buffer should be, the sizes of the relevant .MYI files can be summed up. For InnoDB there is innodb_buffer_pool_size although in this case not only the indices but also the data gets buffered.

3. Further settings

sort_buffer_size (per-thread) is the memory that is used for ORDER BY and GROUP BY. myisam_sort_buffer_size is something completely different and should not be altered.

read_buffer_size (per-thread) is the size of memory chunks that are read from disk into memory at once when doing a full table scan as big tables do not fit into memory completely. This seldomly needs tuning.

Query cache

The main reason not to use any MySQL version below 4.0.1 if you have read-based applications is that beginning with that version, MySQL has the ability to store the result of SELECT queries until their tables are modified.

The Query Cache can be configured with the **query_cache_%** variables. Most important here are the global **query_cache_size** and **query_cache_limit** which prevents single queries with unusual big results larger than this size to use up the whole cache.

Note that the Query Cache blocks have a variable size whose minimum size is query_cache_min_res_unit, so after a complete cache flush the number of free blocks is ideally just one. A large value of Qcache_free_blocks just indicates a high fragmentation.

Worth monitoring are the following variables:

- **Qcache_free_blocks**

  If this value is high it indicates a high fragmentation which does not need to be a bad thing though.

- **Qcache_not_cached**

  If this value is high there are either much uncachable queries (e.g. because they use functions like now()) or the value for query_cache_limit is too low.

- **Qcache_lowmem_prunes**

  This is the amount of old results that have been purged because the cache was full and not because their underlying tables have been modified. query_cache_size must be increased to lower this variable.

Examples:

An empty cache:

```
mysql> SHOW VARIABLES LIKE 'query_cache_type';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_type | ON    |
+------------------+-------+
1 row in set (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 0     |
+------------------+-------+
1 row in set (0.00 sec)

mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Qcache_free_blocks      | 0     |
| Qcache_free_memory      | 0     |
| Qcache_hits             | 0     |
```

```
| Qcache_inserts          | 0     |
| Qcache_lowmem_prunes    | 0     |
| Qcache_not_cached       | 0     |
| Qcache_queries_in_cache | 0     |
| Qcache_total_blocks     | 0     |
+-------------------------+-------+
8 rows in set (0.00 sec)
```

A used cache (savannah.gnu.org):

```
mysql> SHOW VARIABLES LIKE "query_cache_size";
+------------------+----------+
| Variable_name    | Value    |
+------------------+----------+
| query_cache_size | 33554432 |
+------------------+----------+
1 row in set (0.00 sec)
```

```
mysql> SHOW STATUS LIKE "Qcache%";
+-------------------------+----------+
| Variable_name           | Value    |
+-------------------------+----------+
| Qcache_free_blocks      | 1409     |
| Qcache_free_memory      | 27629552 |
| Qcache_hits             | 7925191  |
| Qcache_inserts          | 3400435  |
| Qcache_lowmem_prunes    | 2946778  |
| Qcache_not_cached       | 71255    |
| Qcache_queries_in_cache | 4546     |
| Qcache_total_blocks     | 10575    |
+-------------------------+----------+
8 rows in set (0.00 sec)
```

The matching `my.cnf` configuration parameter is:

```
query_cache_size = 32M
```

To clear the cache (useful when testing a new query's efficiency):

```
mysql> RESET QUERY CACHE;
Query OK, 0 rows affected (0.00 sec)
```

Waiting for locks

The **Table_locks_%** variables show the number of queries that had to wait because the tables they tried to access where currently locked by other queries. These situations can be caused by "LOCK TABLE" statements and also by e.g. simultaneous write accesses to the same table.

Table cache

MySQL needs a certain time just to "open" a table and read its meta data like column names etc.

If many threads are trying to access the same table, it is opened multiple times.

To speed this up the meta data can be cached in the **table_cache** (alias **table_open_cache** since MySQL 5.1.3).

A good value for this setting is the number of max_connections multiplied with the number of usually used tables per SELECT.

Using mysqlreport or by looking at the currently **Open_tables** and ever since **Opened_tables** as well as the **Uptime** the number of necessary table opens per second can be calculated (consider the off-peak times like nights though).

Connections and threads

For every client connection (aka session) MySQL creates a separated thread under the main mysqld process. For big sites with several hundred new connections per second, creating the threads itself can consume a significant amount of time. To speed things up, idle threads can be cached after their client disconnected. As a rule of thumb not more than one thread per second should be newly created. Clients that send several queries to the server should use **persistent connections** like with PHPs mysql_pconnect() function.

This cache can be configured by **thread_cache_size** and monitored with the **threads_%** variables.

To avoid overloads MySQL blocks new connections if more than **max_connections** are currently in use. Start with **max_used_connections** and monitor the number of connection that were rejected in **Aborted_clients** and the ones that timed out in **Aborted_connections**. Forgotten disconnects from clients that use persistent connections can easily lead to a denial of service situation so be aware! Normally connections are closed after **wait_timeout** seconds of being idle.

Temporary tables

It is perfectly normal that MySQL creates temporary tables while sorting or grouping results. Those tables are either be held in memory or if too large be written to disk which is naturally much slower. The number of disk tables among the **Created_tmp_%** variables should be neglectible or else the settings in **max_heap_table_size** and **tmp_table_size** be reconsidered.

Delayed writes

In situations like writing webserver access log files to a database, with many subsequent INSERT queries for rather unimportant data into the same table, the performance can be improved by advising the server to cache the write requests a little while and then send a whole batch of data to disk.

Be aware though that all mentioned methods contradicts ACID compliance because INSERT queries are acknowledged with OK to the client before the data has actually be written to disk and thus can still get lost in case of an power outage or server crash. Additionally the side effects mentioned in the documentation often reads like a patient information leaflet of a modern medicament...

MyISAM tables can be given the **DELAY_KEY_WRITE** option using CREATE or ALTER TABLE. The drawback is that after a crash the table is automatically marked as corrupt and has to be checked/repaired which can take some time.

InnoDB can be told with **innodb_flush_log_at_trx_commit** to delay writing the data a bit. In case of a server crash the data itself is supposed to be still consistent, just the indices have to be rebuilt.

INSERT DELAYED works on main Storage Engines on a per query base.

Further reading

Useful links regarding optimisation of MySQL servers:

- Various newsgroups and the MySQL mailing lists
- A guide to mysqlreport (http://hackmysql.com/mysqlreportguide)
- The book High Performance MySQL (http://www.amazon.com/High-Performance-MySQL-Jeremy-Zawodny/dp/0596003064/)
- Tuning tips from the company EZ (http://ez.no/community/articles/tuning_mysql_for_ez_publish)
- MySysop A php script for mysql optimisation and tuning, demo : MySysop (http://www.fillon.org/mysysop)

References

1. http://dev.mysql.com/doc/refman/5.1/en/optimize-table.html
2. http://dev.mysql.com/doc/refman/5.1/en/datetime.html

APIs

Security

Please, remember that the internet has been created by persons who don't want us to have any sort of secrets. Also remember that a lot of people are payd to learn our secrets and register them somewhere.

Paranoia is a from of intelligence.

Connection parameters

Sometimes, connection parameters (including username and password) are stored in a plain text file, for example a .ini file. This is insecure: if a user guesses how it is called, he can read it. If it's located outside the web server's WWW directory it's more secure, but it's a better practice to store it as a constant in a program file.

It's always possible that a user manages to get your FTP password or other passwords. So the username and the password you use to connect to MySQL should be different from other usernames / passwords.

MySQL passwords must be secure. You don't need to remember them. They should contain lowercase letters, uppercase letters, numbers and symbols (like '_'); they should not contain existing words or your birth date; they should never be sent via email (if they are, there must be some way to modify them); they should not be stored where it is not absolutely necessary to store them.

SQL Injections

What are SQL Injections?

In a perfect world, you would know that values contained in $_POST are values that you can insert into a SQL statement. But in a perfect world there are no poverty or proprietary softwares, so this is not the case. Those values may contain attacks called "SQL Injections". When you expect values like "'42'", you may find values like "'42' OR 1". So, when you try to make a statements like this:

```
DELETE FROM `articles` WHERE `id`=42
```

you may create statements like this instead:

```
DELETE FROM `articles` WHERE `id`=42 OR 1
```

which DELETEs all records.

Also in some cases, you try to make a query like this:

```
SELECT * FROM `my_nice_table` WHERE title='bla bla'
```

And a user may turn it to something like this:

```
SELECT * FROM `my_nice_table` WHERE title='bla bla'; TRUNCATE TABLE `my_nice_table`
```

These are just examples. It's easy to realize if all records are disappeared from your tables. If the tables are properly backed up, you can repopulate them. But there are worst cases. If a user learns how to manipulate your database, he can create an administration account for himself, or he can make modifications to your site's contents that you'll never see, or he can even register payments he has not made.

How to prevent that

Simply, inputs that must represent a value, should not be accepted if they contain something more.

- String values

They are enclosed by 'quotes'. Every quote present in them should be converted into '' or \'. PHP recommends using `mysql_real_escape_string` to substitute these special characters.

- Numbers (integer, float)

They must be numeric input. If they contain something like OR or spaces, they are not numeric.

- Dates

Enclose them within 'quotes' and manage them as if they were strings.

- NULL / UNKNOWN / TRUE /FALSE

These values should never be entered by the user, but should created programmatically.

- SQL names

In some cases, SQL names could be contained in user input. A common case are column names to be used in the ORDER BY clause, which may come from $_GET. Enclose them within `backquotes` and replace every occurrences of ` with ``. Of course, generally speaking, this is a very bad practice if the SQL names are not used ONLY in the ORDER BY clause.

- Comments

User input should never be inserted in SQL comments.

Passwords

When passwords are stored in a database, they are usually encrypted. The encryption should be done by the script and not by MySQL. If it is done via SQL, the passwords are written by the statements as plain text. This means that they are visible through:

- possibly, some system logs, if the communications with the db are done through a network and is not encrypted
- MySQL logs
- SHOW PROCESSLIST

So, one should never send a query like this:

```
SELECT 1 FROM `users` WHERE `password`=MD5('abraxas')
```

But, in PHP, you should write:

```
$sql = "SELECT 1 FROM `users` WHERE `password`=MD5('".md5('abraxas')."')";
```

You should never use insecure encryption functions like PASSWORD(). Also, you should not use 2-way encryption. Only cryptographic hashs, such as SHA256 are secure, and don't use older hash algorithms like MD5.

Passwords, even if they are safely encrypted, should never be retrieved by a SELECT. It's insecure and 1-way encryption does not require that.

SSL

If all contents of your databases are public, there is no reason to use encryption for communications. But generally, this is not the case. Even so, there may be a restricted set of people authorized to submit new content to the site, and this will require the use of passwords.

So often it's a good idea to use SSL encryption. See your driver's documentation to see how to do this (it's always a simple connection option).

Not only will SSL encrypt the network traffic containing the users password, but it can also validates to the user the site as being the correct one using a certificate. One possible attack has a site created to look like the victim site, attempting to get you to submit your username and password.

Optimization

API Calls

Persistent connections

By using persistent connections, we keep the connection with the server open, so that several queries can be executed without the overhead of closing and reopening a connection each time a script is run.

Note that this is not always a good optimization. Try to imagine how many persistent connections a server's RAM should store with a shared hosting setup, if every hosted sites use only persistent connections: there will be too many at once.

Persistent connections are available through many languages.

Free memory

When you execute a query, you get a recordset and put it into a variable. To keep it in memory when you don't need it anymore is a waste of ram. That's why, generally, you should free the memory as soon as possible. If it is possible only few lines before the end of the script, this makes no sense. But in some cases, it is good.

Fetch rows

Many APIs support two ways for fetching the rows: you can put them into a normal array, into an object, or into an associative array. Putting the rows into an object is the slowest way, while putting them into a normal array is the fastest. If you are retrieving a single value per row, putting it into an array may be a good idea.

API vs SQL

Usually, the APIs support some methods which create an SQL statement and send it to the MySQL server. You may obtain the same effects by creating the statement by hand, but it's a slowest way. APIs' methods are generally more optimized.

Reduce client/server communications

- Some scripts use two queries to extract a Pivot table.

Client/server communications are often a bottleneck, so you should try to use only one JOIN instead.

- If you need to use more than one query, you should use only one connection, if possible.
- Only retrieve the fields you really need.
- Try to not include in the SQL command too many meaningless characters (spaces, tabs, comments...).

CREATE ... SELECT, INSERT ... SELECT

When you create a new table from an existing table, you should using CREATE ... SELECT. When you want to populate an existing table from a query, you should use INSERT ... SELECT or a REPLACE ... SELECT. This way, you will tell the server to perform all the needed operations by sending only one SQL statement.

INSERT DELAYED

Many scripts don't check if the INSERTs are successful. If this is the case, you should use INSERT DELAYED instead. So, the client won't wait a confirm from the server before proceeding.

REPLACE

If you run a DELETE and then an INSERT, you need to communicate two SQL commands to the server. Maybe you may want to use REPLACE instead. Possibly, use REPLACE DELAYED.

Other Techniques

Storing data in cookies

Sometimes, session data are stored into a database. This requires at least one UPDATE and one SELECT every time a user loads a page. This can be avoided by storing session data into cookies.

Browsers allow users to not accept cookies, but if they don't accept them, they can't visit many important modern sites.

The only data that can't be securely stored into cookie are passwords. You may set a brief lifetime for cookies though, so the user's privacy is hardly compromised by your cookies. Or you can do the following:

- when a user successfully logs in your site, create a record with CURRENT_TIMESTAMP() and a random ID;
- set a cookie with the ID;
- when the user tries to do something, check if he's logged in:

```
SELECT FROM `access` WHERE `id`=id_from_cookie AND `tstamp`>=CURRENT_TIMESTAMP() - login_lifetime
```

- UPDATE the tstamp

Creating static contents

When a user browses an article or other dynamic contents (which means, contents stored into a database), a HTML document needs to be generated. Often, the page has not variable contents, but just contents which are INSERTed once, and rarely (or never) updated. An article or a list of links are a good example.

So, it may be a good idea creating a program which generates a static HTML page when an article is INSERTed into the database. The page may be deleted and re-generated if the article is UPDATEd. This saves a lot of SQL statements and work for the DBMS.

Of course this requires some privileges which you may not have. If you are using a hosting service, you may need to talk to technical support team about this.

PHP

Drivers

PHP has the following official drivers for MySQL:

- mysql - Older, so it's still used by many web applications; it's a procedural PHP module
- mysqli - faster; can be used as a set of classes or as a normal procedural library
- PDO (PHP Data Objects) - uses PDO, an abstraction layer for interaction with databases which has drivers for MySQL and ODBC.
- PDO_MYSQL support some advanced MySQL features and emulates them if not present.

The functions in the above drivers the extensions recall the methods in the C API. They can use the MySQL Client Library or mysqlnd, a Native Driver for PHP.

Sometimes, enabling both mysql and mysqli may cause some problems; so, if you use only one of them, you should disable the other one.

Also, PHP has a ODBC extension which may be used with MySQL.

PEAR is an important set of PHP classes which supports MySQL.

register_globals and $_REQUEST

PHP has an environment variables called register_globals. Since PHP 4.2 it's set to false by default, and you shouldn't set it. In PHP 5.3 this variable is also deprecated and in PHP 6 has been removed.

However, if your version of PHP supports register_globals, you can verify if it's set to true by calling the function ini_get(). If it's true, thought, you can't modify it with ini_set(). There are two ways to set it off:

- editing php.ini

(impossible if you're using a hosting service)

- adding one line to .htaccess:

```
php_flag register_globals off
```

(sometimes possible in hosting)

The reason is that if register_globals is true, a user can arbitrary add variables to your script by calling them like this:

```
your_script.php?new_variable=new_value
```

You should never user the $_REQUEST superglobal array. It can be used to retrieve variables from:

- $_ENV
- $_GET
- $_POST
- $_COOKIE
- $_SERVER

This is the order followed by PHP (may be modified by the variables_order environment variable). This means that if your script set a server variable called "userid" and you try to read it via $_REQUEST, the user can prevent that by adding a variable to the query string.

Also, you should never blindly trust the validity of HTTP variables.

Debugging

Logging

There a a few ways to debug a MySQL script. For example, if can become necessary to log every SQL request. To do so:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL log_output = 'TABLE';
```

Then it will record every request of the server into the system database `mysql`, table *general_log*.

Exceptions handling

En MySQL, les anomalies du type "division par zéro" ne renvoient pas d'erreur mais NULL.

Toutefois il est possible de lever des exceptions lors des manipulations de table, par exemple pour éviter qu'une liste d'insertions s'arrête au milieu à cause d'une contrainte d'unicité. L'exemple ci-dessous fonctionne sur une table InnoDB (et pas MyISAM)[1] :

```
ALTER TABLE `MaTable1` ADD UNIQUE(`id`);

INSERT INTO MaTable1 (id) VALUES('1');
START TRANSACTION;
      INSERT INTO MaTable1 (id) VALUES('2');
      INSERT INTO MaTable1 (id) VALUES('3');
      INSERT INTO MaTable1 (id) VALUES('1');
IF condition THEN
  COMMIT;
ELSE
  ROLLBACK;
END IF;
```

Ici une erreur surgit lors de la deuxième insertion d'un id=1. Selon une condition, on peut donc annuler les insertions de 2 et 3, ou bien les soumettre.

Errors

1130: Host 'example.com' is not allowed to connect to this MySQL server

Dans le cas d'une connexion depuis un PC distant, le compte utilisé n'est pas autorisé. Il faut donc le configurer avec :

```
GRANT ALL PRIVILEGES ON *.* TO 'utilisateur'@'%' WITH GRANT OPTION;
```

au lieu ou en plus de :

```
GRANT ALL PRIVILEGES ON *.* TO 'utilisateur'@'localhost' WITH GRANT OPTION;
```

1093 - You can't specify target table '...' for update in FROM clause

Cela se produit quand on essaie de mettre à jour ou supprimer des lignes selon une sélection de ces mêmes lignes.

Passer par des `CREATE TEMPORARY TABLE` (voire `DECLARE` si cela rentre dans une variable scalaire).

2003: Can't connect to MySQL server

Changer le paramètre "host".

Erreur : fonctionnalités relationnelles désactivées !

Se produit dans le concepteur de diagramme de phpMyAdmin, il faut l'activer dans config.inc.php.

Invalid use of group function

- Dans le cas d'un `SELECT`, il conviendrait d'utiliser `HAVING` au lieu de `WHERE` pour modifier des enregistrements en fonction d'autres d'une sous-requête.
- Pour un `UPDATE` ou un `DELETE`, les champs comparés par un `IN` ne sont peut-être pas du même type.

SQLSTATE[42000]: Syntax error or access violation

Utiliser phpMyAdmin pour trouver l'erreur de syntaxe.

This version of MySQL doesn't yet support 'LIMIT & IN/ALL/ANY/SOME subquery'

Remplacer les "IN" par des jointures.

CheatSheet

Connect/Disconnect

```
mysql -h <host> -u <user> -p<passwd>
mysql -h <host> -u <user> -p
Enter password: ********
mysql -u user -p
mysql
mysql -h <host> -u <user> -p <Database>
```

Query

```
SELECT * FROM table
SELECT * FROM table1, table2, ...
```

```
SELECT field1, field2, ... FROM table1, table2, ...
SELECT ... FROM ... WHERE condition
SELECT ... FROM ... WHERE condition GROUP BY field
SELECT ... FROM ... WHERE condition GROUP BY field HAVING condition2
SELECT ... FROM ... WHERE condition ORDER BY field1, field2
SELECT ... FROM ... WHERE condition ORDER BY field1, field2 DESC
SELECT ... FROM ... WHERE condition LIMIT 10
SELECT DISTINCT field1 FROM ...
SELECT DISTINCT field1, field2 FROM ...

SELECT ... FROM t1 JOIN t2 ON t1.id1 = t2.id2 WHERE condition
SELECT ... FROM t1 LEFT JOIN t2 ON t1.id1 = t2.id2 WHERE condition
SELECT ... FROM t1 JOIN (t2 JOIN t3 ON ...) ON ...
SELECT ... FROM t1 JOIN t2 USING(id) WHERE condition
```

Conditionals

```
field1 = value1
field1 <> value1
field1 LIKE 'value _ %'
field1 IS NULL
field1 IS NOT NULL
field1 IN (value1, value2)
field1 NOT IN (value1, value2)
condition1 AND condition2
condition1 OR condition2
```

Data Manipulation

```
INSERT INTO table1 (field1, field2, ...) VALUES (value1, value2, ...)
INSERT table1 SET field1=value_1, field2=value_2 ...

LOAD DATA INFILE '/tmp/mydata.txt' INTO TABLE table1
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'

DELETE FROM table1 / TRUNCATE table1
DELETE FROM table1 WHERE condition
-- join:
DELETE FROM table1, table2 WHERE table1.id1 = table2.id2 AND condition

UPDATE table1 SET field1=new_value1 WHERE condition
-- join:
UPDATE table1, table2 SET field1=new_value1, field2=new_value2, ...
WHERE table1.id1 = table2.id2 AND condition
```

Browsing

```
SHOW DATABASES
SHOW TABLES
SHOW FIELDS FROM table / SHOW COLUMNS FROM table / DESCRIBE table / DESC table / EXPLAIN table
SHOW CREATE TABLE table
SHOW CREATE TRIGGER trigger
SHOW TRIGGERS LIKE '%update%'
SHOW PROCESSLIST
KILL process_number
SELECT table_name, table_rows FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = '**yourdbname**';
$ mysqlshow
$ mysqlshow database
```

Create / delete / select / alter database

```
CREATE DATABASE [IF NOT EXISTS] mabase [CHARACTER SET charset] [COLLATE collation]
CREATE DATABASE mabase CHARACTER SET utf8
DROP DATABASE mabase
USE mabase

ALTER DATABASE mabase CHARACTER SET utf8
```

Create/delete/modify table

```
CREATE TABLE table (field1 type1, field2 type2, ...)
CREATE TABLE table (field1 type1 unsigned not null auto_increment, field2 type2, ...)
CREATE TABLE table (field1 type1, field2 type2, ..., INDEX (field))
CREATE TABLE table (field1 type1, field2 type2, ..., PRIMARY KEY (field1))
CREATE TABLE table (field1 type1, field2 type2, ..., PRIMARY KEY (field1, field2))
CREATE TABLE table1 (fk_field1 type1, field2 type2, ...,
  FOREIGN KEY (fk_field1) REFERENCES table2 (t2_fieldA)
    [ON UPDATE] [CASCADE|SET NULL|RESTRICT]
    [ON DELETE] [CASCADE|SET NULL|RESTRICT])
CREATE TABLE table1 (fk_field1 type1, fk_field2 type2, ...,
  FOREIGN KEY (fk_field1, fk_field2) REFERENCES table2 (t2_fieldA, t2_fieldB))
CREATE TABLE table IF NOT EXISTS (...)

CREATE TABLE new_tbl_name LIKE tbl_name
  [SELECT ... FROM tbl_name ...]

CREATE TEMPORARY TABLE table (...)

CREATE table new_table_name as SELECT [ *|column1, column2 ] FROM table_name

DROP TABLE table
DROP TABLE IF EXISTS table
DROP TABLE table1, table2, ...
DROP TEMPORARY TABLE table

ALTER TABLE table MODIFY field1 type1
ALTER TABLE table MODIFY field1 type1 NOT NULL ...
ALTER TABLE table CHANGE old_name_field1 new_name_field1 type1
ALTER TABLE table CHANGE old_name_field1 new_name_field1 type1 NOT NULL ...
ALTER TABLE table ALTER field1 SET DEFAULT ...
```

```
ALTER TABLE table ALTER field1 DROP DEFAULT
ALTER TABLE table ADD new_name_field1 type1
ALTER TABLE table ADD new_name_field1 type1 FIRST
ALTER TABLE table ADD new_name_field1 type1 AFTER another_field
ALTER TABLE table DROP field1
ALTER TABLE table ADD INDEX (field);
ALTER TABLE table ADD PRIMARY KEY (field);

-- Change field order:
ALTER TABLE table MODIFY field1 type1 FIRST
ALTER TABLE table MODIFY field1 type1 AFTER another_field
ALTER TABLE table CHANGE old_name_field1 new_name_field1 type1 FIRST
ALTER TABLE table CHANGE old_name_field1 new_name_field1 type1 AFTER another_field

ALTER TABLE old_name RENAME new_name;
```

Keys

```
CREATE TABLE table (..., PRIMARY KEY (field1, field2))
CREATE TABLE table (..., FOREIGN KEY (field1, field2) REFERENCES table2 (t2_field1, t2_field2))
ALTER TABLE table ADD PRIMARY KEY (field);
ALTER TABLE table ADD CONSTRAINT constraint_name PRIMARY KEY (field, field2);
```

create/modify/drop view

```
CREATE VIEW view AS SELECT ... FROM table WHERE ...
```

Privileges

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

GRANT ALL PRIVILEGES ON base.* TO 'user'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT, INSERT, DELETE ON base.* TO 'user'@'localhost' IDENTIFIED BY 'password';
REVOKE ALL PRIVILEGES ON base.* FROM 'user'@'host'; -- one permission only
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'user'@'host'; -- all permissions

SET PASSWORD = PASSWORD('new_pass')
SET PASSWORD FOR 'user'@'host' = PASSWORD('new_pass')
SET PASSWORD = OLD_PASSWORD('new_pass')

DROP USER 'user'@'host'
```

Main data types

```
TINYINT   (1o: -127+128)
SMALLINT  (2o: +-65 000)
MEDIUMINT (3o: +-16 000 000)
INT       (4o: +-2 000 000 000)
BIGINT    (8o: +-9.10^18)
  Precise interval: -(2^(8*N-1)) -> (2^8*N)-1
  /!\ INT(2) = "2 digits displayed" -- NOT "number with 2 digits max"
```

```
INT NOT NULL auto_increment PRIMARY KEY -- auto-counter for PK
```

```
FLOAT(M,D) DOUBLE(M,D) FLOAT(D=0->53)
  /!\ 8,3 -> 12345,678 -- NOT 12345678,123!
```

```
TIME (HH:MM) YEAR (AAAA) DATE (AAAA-MM-JJ) DATETIME (AAAA-MM-JJ HH:MM; années 1000->9999)
  TIMESTAMP (like DATETIME, but 1970->2038, compatible with Unix)
```

```
VARCHAR (single-line; explicit size)
TEXT (multi-lines; max size=65535)
BLOB (binary; max size=65535)
  Variants for TEXT&BLOB: TINY (max=255) MEDIUM (max=~16000) LONG (max=4Go)
 Ex: VARCHAR(32), TINYTEXT, LONGBLOB, MEDIUMTEXT
```

```
ENUM ('value1', 'value2', ...) -- (default NULL, or '' if NOT NULL)
```

Forgot root password?

```
$ /etc/init.d/mysql stop
$ mysqld_safe --skip-grant-tables &
$ mysql # on another terminal
mysql> UPDATE mysql.user SET password=PASSWORD('nouveau') WHERE user='root';
## Kill mysqld_safe from the terminal, using Control + \
$ /etc/init.d/mysql start
```

Repair tables after unclean shutdown

```
mysqlcheck --all-databases
mysqlcheck --all-databases --fast
```

Loading data

```
mysql> SOURCE input_file
$ mysql database < filename-20120201.sql
$ cat filename-20120201.sql | mysql database
```

Contributors

- Beuc: structured the book in chapters and setup the print version; wrote the initial Administration, Database Manipulation, CheatSheet section; contributed to Introduction (MySQL license), Optimization (query cache and benchmark examples, indices exercise), Table types (reference other possible table types), Language (datetime/timestamp valid intervals), Pivot table (alternate version w/o maths). I'd like to thank my employer, Cliss XXI (http://www.cliss21.com), for giving me time to work on these chapters. Then wrote the Replication section (on free time).
- JackPotte: added a certain amount of data on several pages, and tested every script on MySQL 5.6 when translating into the French Wikibooks.
- Lathspell: wrote the initial Optimization section
- LucienPetit: wrote the initial *OpenOffice Base and ODBC* section. I'd like to thank my employer, Cliss XXI (http://www.cliss21.com), for giving me time to work on it (but I also worked on my free time).
- Shantanuo: wrote the initial Language section.
- Sante Caserio: started Stored Programs; started APIs; edited some existing stuff; added Language.Operators; added Table Types.Metadata about Sorage Engines;

- http://stackoverflow.com/questions/2950676/difference-between-set-autocommit-1-and-start-transaction-in-mysql-have-i-misse