Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database

MCS 275 Lecture 25
Programming Tools and File Management
Jan Verschelde, 8 March 2017
1 MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2 Database for Python scripts
   - problem statement
   - create database and table

3 Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
Starting and Stopping the Daemon

It may be that MySQL is started at boot time.

Otherwise:

$ sudo mysqld_safe

Starting mysqld daemon with databases from /usr/local/mysql/data

Shutting the MySQL server down:

$ sudo mysqladmin shutdown
Creating and Deleting Databases

To create a database we use the `create` command with `mysqladmin`:

```
$ sudo mysqladmin create mydb
```

To delete the database we use the `drop` command with `mysqladmin`:

```
$ sudo mysqladmin drop mydb
```

Dropping the database is potentially a very bad thing to do. Any data stored in the database will be destroyed.

Do you really want to drop the 'mydb' database [y/N] y
Database "mydb" dropped
user administration

Running `mysql` as `sudo mysql` or `mysql -u root`, we give privileges to certain users to some databases.

First we have to create a user account in `mysql`. For example, for a user `name` on localhost:

```sql
mysql> create user 'name'@'localhost';
```

There is the possibility to set a password.

To grant user with `name` all privileges on all databases:

```sql
mysql> GRANT ALL PRIVILEGES ON *.* TO 'name'@'localhost';
```

Replacing the wild card `*` by specific names of databases and/or tables allows to restrict the privileges to certain databases and/or tables.
Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
running the monitor mysql

$ mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.29 MySQL Community Server (GPL)

Copyright (c) 2000, 2012, Oracle and/or its affiliates. All

Oracle is a registered trademark of Oracle Corporation and/affiliates. Other names may be trademarks of their respective

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

mysql>
leaving mysql

mysql> exit
Bye

Command line MySQL is very useful to try single commands.

Careful programming:

1. first try a command before placing it into a script,
2. have the script print the command,
3. execute one printed command as a test.
**Basic *mysql* Commands**

<table>
<thead>
<tr>
<th>command syntax</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>use</strong> &lt;dbname&gt;</td>
<td>make database current</td>
</tr>
<tr>
<td><strong>show tables</strong></td>
<td>show tables in current database</td>
</tr>
<tr>
<td><strong>create table</strong> &lt;name&gt; &lt;field(s)&gt;</td>
<td>create a table</td>
</tr>
<tr>
<td><strong>explain</strong> &lt;name&gt;</td>
<td>display data types of all fields</td>
</tr>
<tr>
<td><strong>drop table</strong> &lt;name&gt;</td>
<td>delete a table</td>
</tr>
</tbody>
</table>

To change a table, we use *queries*:

<table>
<thead>
<tr>
<th>command syntax</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>select</strong> &lt;field(s)&gt; <strong>from</strong> &lt;table&gt;</td>
<td>retrieve records</td>
</tr>
<tr>
<td><strong>insert into</strong> &lt;table&gt; <strong>&lt;values&gt;</strong></td>
<td>insert records</td>
</tr>
<tr>
<td><strong>delete from</strong> &lt;table&gt;</td>
<td>delete records</td>
</tr>
<tr>
<td><strong>update</strong> &lt;table&gt; <strong>set</strong> &lt;values&gt;</td>
<td>update records</td>
</tr>
</tbody>
</table>

In addition:  
**where** <criteria>  
**order by** <field> **ASC | DSC**
Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
Database to manage our Scripts

So far, there are over 70 Python scripts posted at the course web site.

The scripts are listed chronologically grouped along the lectures . . . but then, there are also the scripts for the projects and quizzes.

Goal: build a systematical catalog.
→ sort the scripts in several ways
Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
creating a database and granting access

A database is created by the administrator `mysqladmin`:

$ sudo mysqladmin create OurPyFiles

To grant access to users who are not sudoers:

$ sudo mysql

```
mysql> GRANT ALL ON OurPyFiles.* TO 'jan'@'localhost';
Query OK, 0 rows affected (0.05 sec)
```

mysql> Bye

$ mysql

```
mysql> use OurPyFiles;
Database changed
```
The database **OurPyFiles** will have one table: **scripts**.

For every script, we have a type (L, P, or Q), a number, a date, and a file name.

```sql
mysql> use OurPyFiles
Database changed
mysql> create table scripts
    
    -> (t CHAR(1), n INT, d DATE, f CHAR(20));
Query OK, 0 rows affected (0.00 sec)
```

**mysql** commands are closed with a semicolon ;
to see the data types in a table

mysql> explain scripts;
+-----------------+-------+-----+-----------------+-----------------+-------+
| Field | Type   | Null | Key  | Default         | Extra |
+-----------------+-------+-----+-----------------+-----------------+-------+
| t    | char(1)| YES  |     | NULL            |       |
| n    | int(11)| YES  |     | NULL            |       |
| d    | date   | YES  |     | NULL            |       |
| f    | char(20)| YES |     | NULL            |       |
+-----------------+-------+-----+-----------------+-----------------+-------+
4 rows in set (0.10 sec)
**data types in MySQL**
Most commonly used data types:

<table>
<thead>
<tr>
<th>numeric types</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>integer in $[-2^{31}, 2^{31} - 1]$</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>2-byte integer</td>
</tr>
<tr>
<td>FLOAT</td>
<td>floating-point number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>date and time</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>year as $yyyy$</td>
</tr>
<tr>
<td>DATE</td>
<td>date in format $yyyy-mm-dd$</td>
</tr>
<tr>
<td>TIME</td>
<td>time in format $hh:mm:ss$</td>
</tr>
<tr>
<td>DATETIME</td>
<td>$yyyy-mm-dd$ $hh:mm:ss$</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>date expressed in seconds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>string types</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(size)</td>
<td>string of length $size$</td>
</tr>
<tr>
<td>TEXT</td>
<td>strings of unlimited length</td>
</tr>
</tbody>
</table>
Seeing Tables and their Fields

mysql> show tables;
+----------------------+
| Tables_in_ourpyfiles |
+----------------------+
| scripts |
+----------------------+
1 row in set (0.00 sec)

mysql> explain scripts;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| t     | char(1)  | YES  |     | NULL    |       |
| n     | int(11)  | YES  |     | NULL    |       |
| d     | date     | YES  |     | NULL    |       |
| f     | char(20) | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
4 rows in set (0.00 sec)
Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
inserting records with a Python script

The table will contain over 70 records.

Entering the data manually is tedious and may lead to errors.

Therefore, we use MySQLdb, or pymysql with Python 3.6

```
$ python3
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information
>>> import pymysql
```

Two tasks:

1. grab the headers of each .py file
2. insert header fields into table
using the `os` module to scan files

All Python scripts are in some directory on disk.

With the `os` module we write utilities which are independent of the operating system.

Commands we will use:

- `os.listdir(<directory>)` returns a list of strings of names of files and directories in `<directory>`
- `os.getcwd()` returns the path name of the current working directory
- `os.chdir(<path name>)` changes the current working directory to the `<path name>`.
Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
grabbing the headers

Every script is documented in a uniform manner:

```
# L-25 MCS 275 Wed 8 Mar 2017 : grabpyhead.py

Grabs the header line of all .py programs. When the file starts with '#!', the header line is not the first but the second line. Every file has four fields: type, number, date, and file name.

Format conversion:

('L', '25', 'Wed 8 Mar 2017', 'grabpyhead.py')
```
import os

def has_py_ext(name):
    """
    Returns True if the name ends in ".py", returns False otherwise.
    """
    try:
        return name[-3:] == ".py"
    except:
        return False
def start_path(line):
    
    Returns True if the line appears to be the path of the python interpreter, returns False otherwise.
    
    try:
        return line[0:2] == '#!'
    except:
        return False
def split_fields(line):
    ""
    Returns a tuple with the fields of the header.
    ""
    L0 = line.split(' : ')
    fname = L0[1]
    L1 = L0[0].split(' MCS 275 ')
    fdate = L1[1]
    L2 = L1[0].split(' ')
    L3 = L2[1].split('-')
    ftype = L3[0]
    fnumb = L3[1]
    return (ftype, fnumb, fdate, fname)
def enum_fields(d, fun):
    
    Enumerates all fields in the header of the .py files in the directory d. For each field, the function f is called. Returns the number of .py files.
    
    L = os.listdir(d)
cnt = 0
for filename in L:
    if has_py_ext(filename):
        cnt = cnt + 1
        file = open(filename, 'r')
        line = file.readline()
After `line = file.readline()`:

```python
    if start_path(line):
        line = file.readline()
    try:
        fun(split_fields(line[:-1]))  # omit \n
    except:
        print('exception occurred with file ',\
              filename)
    file.close()
return cnt
```
callback functions in iterators

The `enum_fields` routine is an *iterator*.

An iterator enumerates all items in a collection.

Actions on the items are of no concern of the iterator, display items, select items, or . . .

Good software design:

1. write iterator separately with simple callback function (e.g.: `print`) to test
2. make callback functions specific for applications
def print_fields(data):
    
    Use as argument to test the enumerator.
    
    print(data)

def main():
    
    Prints the header of all .py files in the current directory.
    
    nbr = enum_fields('.', print_fields)
    print('counted %d .py files' % nbr)

if __name__ == '__main__':
    main()
Working with MySQL

1. MySQL basics
   - starting and stopping the daemon
   - running the monitor mysql

2. Database for Python scripts
   - problem statement
   - create database and table

3. Using MySQLdb or pymysql
   - inserting records with Python script
   - scanning files and grabbing the headers
   - filling the table in the database
Using the pymysql API

Five things to remember:

1. import pymysql

2. connect to the database:
   `<connection> = pymysql.connect(db="<dbname>")`

3. create a cursor object:
   `<cursor> = <connection>.cursor()`

4. execute mysql commands:
   `<cursor>.execute(<command string>)`
   returns number of rows in the result

5. retrieving results of queries:
   `<cursor>.fetchone()` returns single row
   `<cursor>.fetchall()` returns all rows
   `<cursor>.rowcount` returns number of rows
Converting Date Formats

```python
import os
import pymysql
from grabpyhead import enum_fields

def my_date(data):
    """
    Converts a string such as Wed 9 Mar 2016 into the format 2016-03-09.
    """

    months = {"Jan":"01", "Feb":"02", "Mar":"03", \
              "Apr":"04", "May":"05", "Jun":"06", \
              "Jul":"07", "Aug":"08", "Sep":"09", \
              "Oct":"10", "Nov":"11", "Dec":"12"}
    vals = data.split(' ')
    day = '%02d' % int(vals[1])
    return vals[3] + '-' + months[vals[2]] + '-' + day
```
def insert_data(cur, doit=False):
    ""
    Data is inserted into the database, using the cursor cur.
    ""

def insert(data):
    ""
    Uses the tuple data to insert into the table scripts.
    ""
    cmd = 'insert into scripts values ('
    cur.execute(cmd)
```
def insert_data(cur, doit=False):
    """
    Data is inserted into the database, using the cursor cur.
    """

def insert(data):
    ... code omitted ...
    cur.execute(cmd)

    if doit:
        nbr = enum_fields('.', insert)
    else:
        nbr = enum_fields('.', print)
    return nbr
```
filling the table, the `main()` in `filldb.py`

def main():
    ""
    Prints the header of all .py files in the current directory.
    """

    pth = os.getcwd()
    os.chdir('../MCS275py')
    # db = MySQLDb.connect(db="OurPyFiles")
    ourdb = pymysql.connect(db="OurPyFiles")
    crs = ourdb.cursor()
    ans = input("really do it ? (y/n) ")
    nbr = insert_data(crs, ans == 'y')
    print('inserted %d .py files' % nbr)
    ourdb.commit()
    os.chdir(pth)
Some Queries

Sort by date $d$:

```
mysql> select * from scripts order by d;
```

Recall that $*$ is a wild card, the returned table contains all fields of `scripts`.

To retrieve date and file name of all quiz scripts:

```
mysql> select d, f from scripts where t = "Q";
```
import pymysql

def main():
    """
    Executes a simple query to the database.
    """
    # db = MySQLdb.connect(db="OurPyFiles")
    db = pymysql.connect(db="OurPyFiles")
    c = db.cursor()
    q = 'select * from scripts order by d'
    lc = c.execute(q)
    print('found %d rows' % int(lc))
    while True:
        print(c.fetchone())
        ans = input('see more ? (y/n) ')
        if ans != 'y': break
Summary and Exercises


1. Extend the script `filldb.py` so that it also recursively looks for `.py` files in all subdirectories.

2. The database is not normalized because type, number, and dates are redundant. Use `mysql` to select from `scripts` to create a table `typedates` to store only data like "L-25" and "8 Mar 2017". Select from `scripts` to create a table `typefiles` to store "L-25" and "viewdbdata.py".

3. Modify the Python script `filldb.py` so it fills the tables `typedates` and `typefiles` (defined in the previous exercise) while scanning the `.py` files.

4. Extend the script `viewdbdata.py` asking for sorting directives. There are 32 possible orders.