**11/20/2019**

**Introduction to Python**

by K. Yue

**1. Resources**

* Python: <https://www.python.org/> (usually latest stable supported version)
* Version 3.5:
  + Manual: <http://docs.python.org/3.7/reference/index.html>
  + Tutorial: <https://docs.python.org/3.7/tutorial/interpreter.html>
  + Standard library: <https://docs.python.org/3.7/library/index.html>
* A good whirlwind tour of Python: <https://learnxinyminutes.com/docs/python3/>
* Python's Beginner Guide for programmers: <https://wiki.python.org/moin/BeginnersGuide/Programmers>
* Style Guide for Python Code: <https://www.python.org/dev/peps/pep-0008/>
* You are recommended to install Python using customization to put Python in the top level of your C drive. For example: in c:\Python35-32\.

Python shell: REPL

**2. Basics**

* Python is a (very) high-level, open source, general-purpose, object-oriented, extensible, interpreted scripting programming language.
* Python is a scripting language:
  1. Especially good for scripting and rapid application development (relative easy to write; shorter programs)
  2. Object-oriented language (more so than many other OO languages)

Java: int I;   
I is not an object, predefined data type.

* 1. Dynamic typing: an object bound to a variable can change type dynamically.

>>> i=5

>>> i

5

>>> id(i)

1355888464

>>> i='hello'

>>> i

'hello'

>>> id(i)

52840192

* 1. Strongly typed: less implicit type conversion.
  2. Native high level data structures: list, tuple, set, dictionary, etc.
  3. Interpreted
  4. Automatic garbage collection
  5. Open source
  6. Use good software engineering principle

**Python programs**

* A Python program contains *logical lines*.
* A logical line usually contain one *physical line*, but can be extended by '\' to contain many physical lines.
* Implicit line joining is performed for (), [] and {} expressions. It is preferred to using '\'.
* Do not use ';' as a statement separator.
* Indentation (white spaces at the beginning of lines) has meaning. They can be mandatory.
* *Indentation level* is used to group statements.
* Python 2: do not mix tabs and spaces.
* Python 3:
  + Disallow mixing of tabs and spaces.
  + Spaces are preferred.
* Comments start with a '#'.

**Running Python in Windows**

* To start the Python interpreter, open a command line prompt and execute:

set path=%path%;C:\Python35-32  
python

* You may need to replace "C:\Python35-32" by the location of your python installation.
* Use Control-z or quit() to exit Python interpreter.
* To run a Python program, helloworld.python, use:

python helloworld.py

**How to learn a new language:**

1. syntax
2. concepts that are familiar to you: notice any difference.
3. concepts that are new to you
4. resources and libraries
5. design patterns and best practices

**Some basics to get you interested (hopefully):**

* No ;
* No i++
* Multiple assignment: a,b = b,a
* strings are automatically concatenated: print("ab" "cd")
* strings are automatically indexed:
  + a='12345'
  + print(a[1:3])
* id() to find out id of an object.
* indentation is mandatory, not optional.
* arrays are much more general.

**Example of concepts that may be new to you:**

* Immutable or mutable objects?
  + The *value* of an immutable object cannot be changed.
  + Use the id() function for experimentation
* Automatic garbage collection: "Objects are never explicitly destroyed; however, when they become unreachable they may be garbage-collected."

***Example:***

Interpreter:

C:\Users\yue>python  
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> hello\_world = 'hello world, from ITEC 3335'  
>>> print(hello\_world)  
hello world, from ITEC 3335  
>>> exit()

Print first 100 Fibonacci numbers:

#   print first 100 fibonacci numbers  
a, b, count = 0, 1, 1  
while count <= 100:  
   print(b, ' ', end='')  
   a, b, count = b, a+b, count+1  
print()

Note the use of multiple assignment statements. In other languages, such as Java, you may need to replace:

a, b, count = b, a+b, count+1

by

temp=b;  
b=a+b;  
a=temp;  
count++;

Other examples as will be discussed in the class.

***Additional Material: Example: A Non-trivial program***

Consider the following weather information file: [201401daily\_sample.csv](http://dcm.uhcl.edu/yue/courses/itec3335/current/notes/python/201401daily_sample.csv). Write a Python program to read the 'codeSum' column (#23) and output the count of each codeSum. Note that the codeSum columns may contain more than one codeSum separated by white spaces. Example:

SN FG+ FZFG BR UP

has five codeSum:

* SN
* FG+
* FZFG
* BR
* UP

codeSum.py: more Java-like style

import sys, getopt  
import re  
from operator import itemgetter  
  
#   getopt: C-style parser for command line options.  
#   sys: System-specific parameters and functions.  
#   re: regular expression operation  
#   operator: methods for built-in operators.  
#       (useful when passing the function as a parameter.)  
  
#   Read and process a weather information file.  
#   It parse the codeSum column (#23) in the CSV file  
#   and show the count of each codeSum.  
  
f = open(sys.argv[1], 'r')  
result = [];  
heading = f.readline().split(',')  
num\_line = 0;  
  
for line in f:  
    line = line.rstrip()    #   strip trailing white spaces  
    #   result is an array of arrays:  
    #   Add the array contains columns of the current line  
    #   to result.  
    result.append(line.split(','))  
    num\_line = num\_line + 1  
f.close()  
  
#   Debug:  
#   for i in range(len(result)):  
#       print (str(i) + ":" + str(result[i][22]))  
  
#   count is a dictionary with the key being the individual CodeSummary  
count = {};  
for i in range(len(result)):  
    #   process one reading.  
    line = result[i][22].strip()  
    if line:  
        #   Get all codeSummary and update their counts.  
        summary = re.split('\s+', line)  
        for j in range(len(summary)):  
            #   Debug:  
            #   print (str(j) + ":" + str(summary[j]) + "---")  
            if summary[j] in count.keys():  
                count[summary[j]] += 1  
            else:  
                count[summary[j]] = 1  
  
#   Print result in the sorted order of codeSum.                 
for key, value in sorted(count.items(), key=itemgetter(0)):  
    print (key + ": " + str(value))  
  
             
Running the program:

>...python codeSum.py 201401daily\_sample.csv  
BR: 18  
FG: 2  
FG+: 5  
FZFG: 8  
HZ: 10  
RA: 2  
SN: 14  
UP: 4  
  
If you are really Python curious, a second, more Python-like version:

from collections import defaultdict  
import sys, getopt  
import re  
from operator import itemgetter  
  
#   Read and process a weather information file.  
#   It parse the codeSum column (#23) in the CSV file  
#   and show the count of each codeSum.  
  
f = open(sys.argv[1], 'r')  
heading = f.readline().split(',')  
result = [line.strip().split(',') for line in f.readlines()]  
f.close()  
  
#   count is a dictionary with the key being the individual CodeSummary  
count = defaultdict(int)  
for field in filter(lambda a: a, map(lambda w: w[22].strip(), result)):  
   for sym in re.split('\s+', field):  
      count[sym] += 1  
    
#   Print result in the sorted order of codeSum.                 
for key, value in sorted(count.items(), key=itemgetter(0)):  
    print (key + ": " + str(value))

**MySQL Programming in Python**

by K. Yue

**1. MySQL Drivers for Python**

* There are many MySQL drivers for Python. Some examples:
  + MySQLDB Python: <http://sourceforge.net/projects/mysql-python/>
  + MySQL Connector Python: <https://dev.mysql.com/downloads/connector/python/>: written with pure Python.
  + MySQL Connector Python's Developer Guides: <http://dev.mysql.com/doc/connector-python/en/index.html>
* Selecting the right driver is important and not straight forward. Examples:
  + An example comparing driver performance: [https://github.com/Benoss/PythonMysqlDriversTes](https://github.com/Benoss/PythonMysqlDriversTest)t
  + PyMySQL evaluation: <https://wiki.openstack.org/wiki/PyMySQL_evaluation>
* Because of the problem of MySQL Connector in Python 3, you should use PyMySQL.

**2. MySQL Driver**

**PyMySQL:**

* Installation:
  + Install Pip for Python, if necessary: <https://pip.pypa.io/en/stable/installing/>.
  + Command: "pip install PyMySQL"
* Documentation: <http://pymysql.readthedocs.io/en/latest/>
* Support: <https://stackoverflow.com/questions/tagged/pymysql>

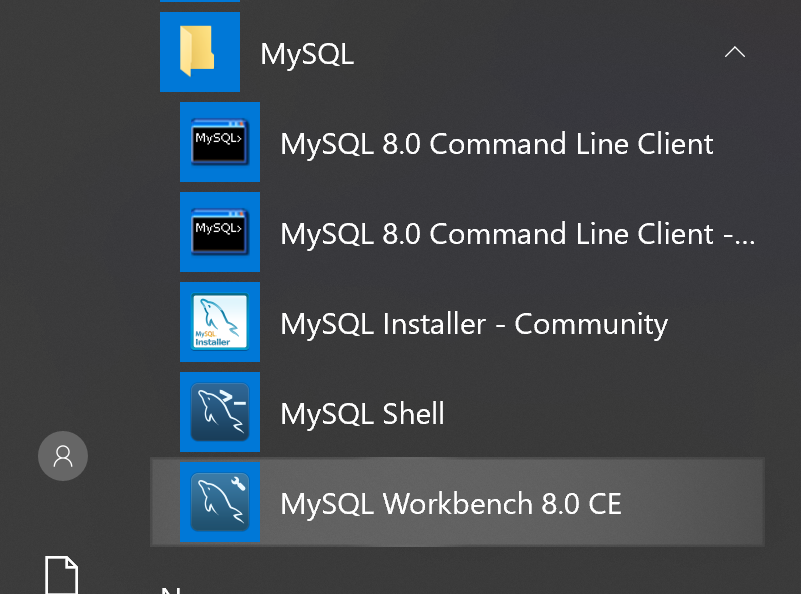
**Additional Materials: MySQL Connector Python (not used in this course, but it is a good resource)**

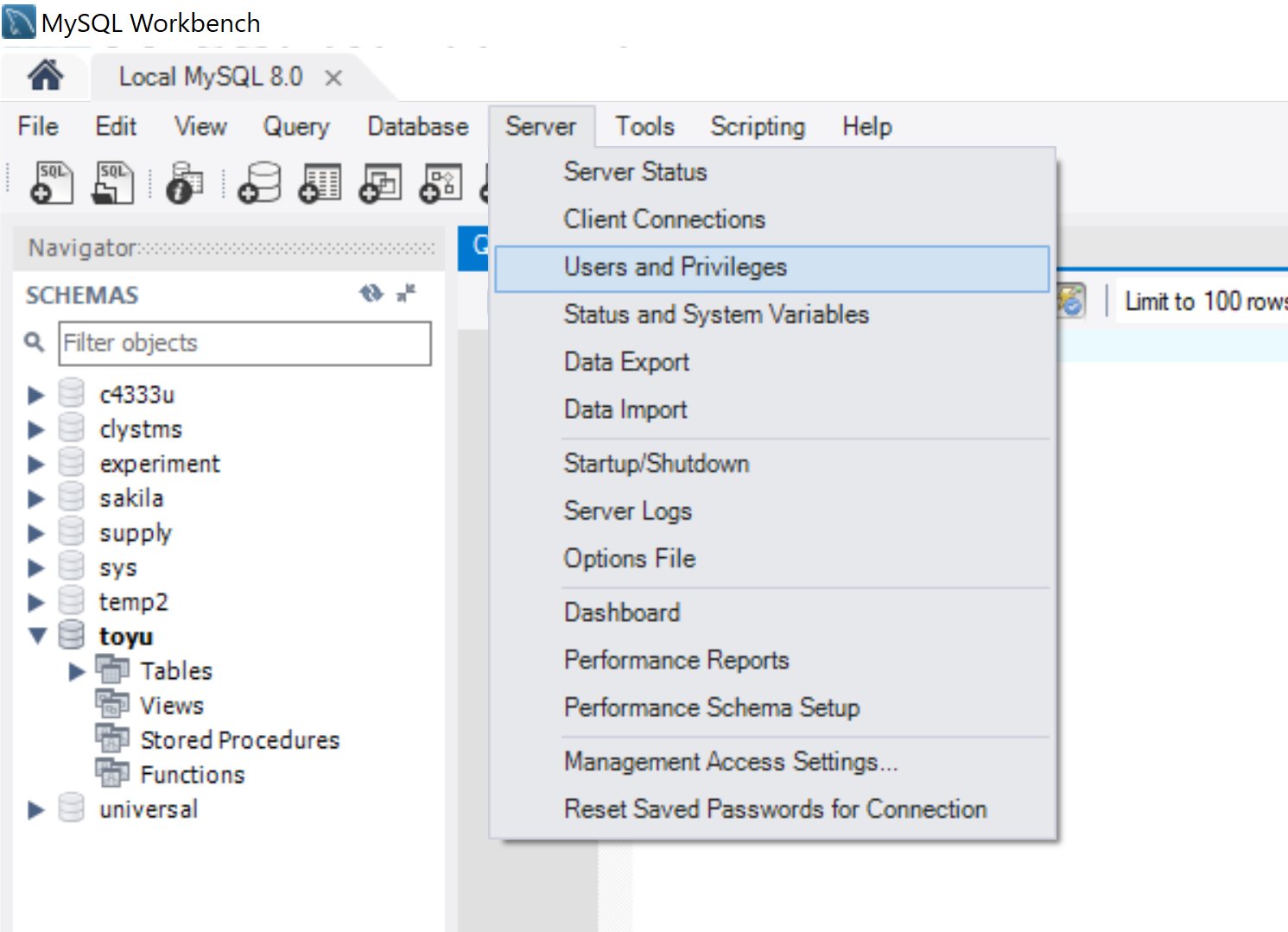
* API reference: <http://dev.mysql.com/doc/connector-python/en/connector-python-reference.html>
* Read MySQL Connector Python's Guideline for developer: [http://dev.mysql.com/doc/connector-python/en/connector-python-coding.htm](http://dev.mysql.com/doc/connector-python/en/connector-python-coding.html)
* Examples of good tips and insight:
  1. Use config.py module to store database connection information."
  2. "Any application that accepts input must expect to handle bad data."
  3. "Data that you choose to store in MySQL instead is likely to have special characteristics."
  4. "you can use Python's triple-quoting mechanism to enclose the entire statement."
  5. "Oracle recommends the ENGINE=INNODB clause for most tables, and makes InnoDB the default storage engine in MySQL 5.5 and up."
* Compliant with Python Database API Specification v2.0: <https://www.python.org/dev/peps/pep-0249/#module-interface>.

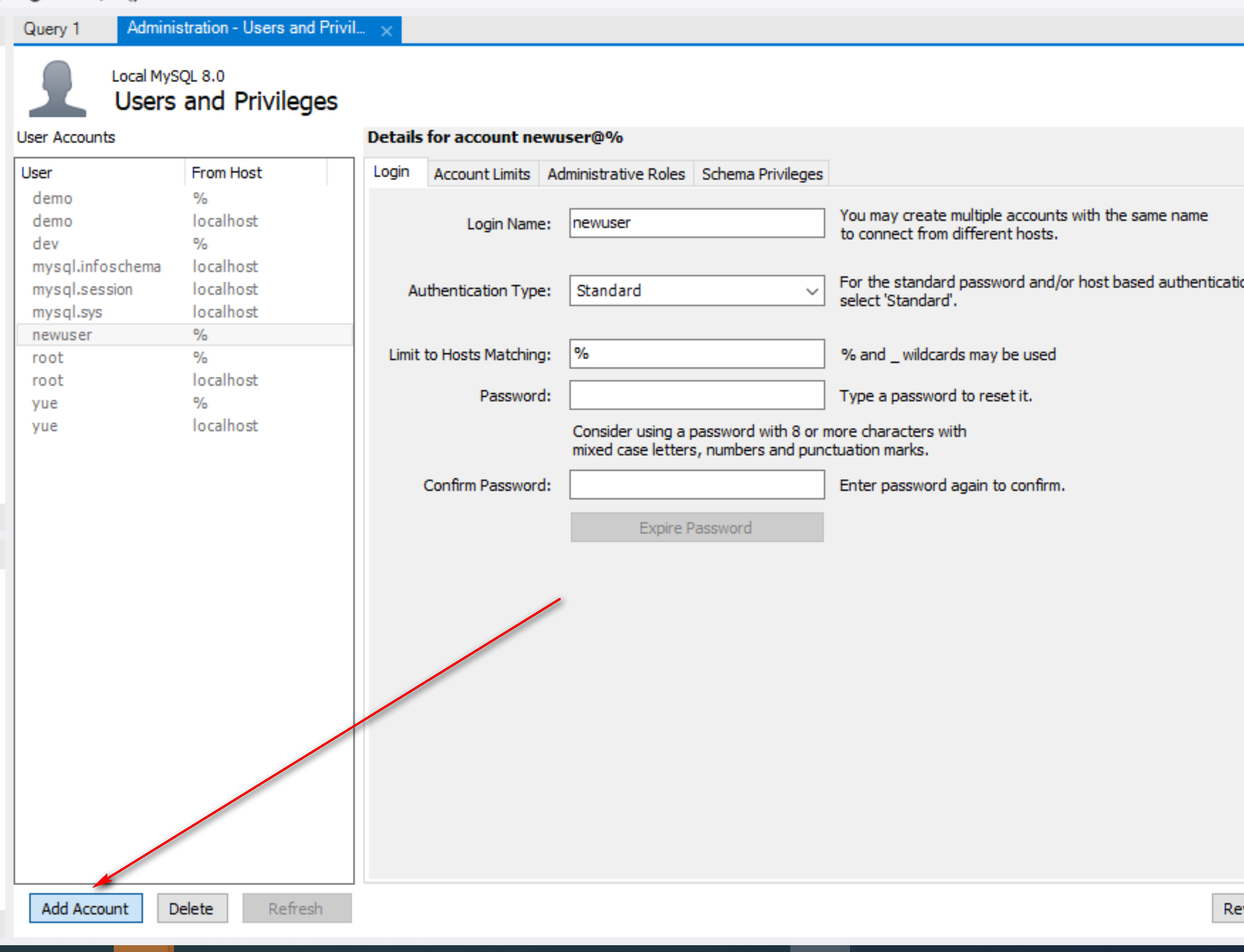
**Python Programming with MySQL**

Create a user account:

1/ Run MYSQL Workbench:

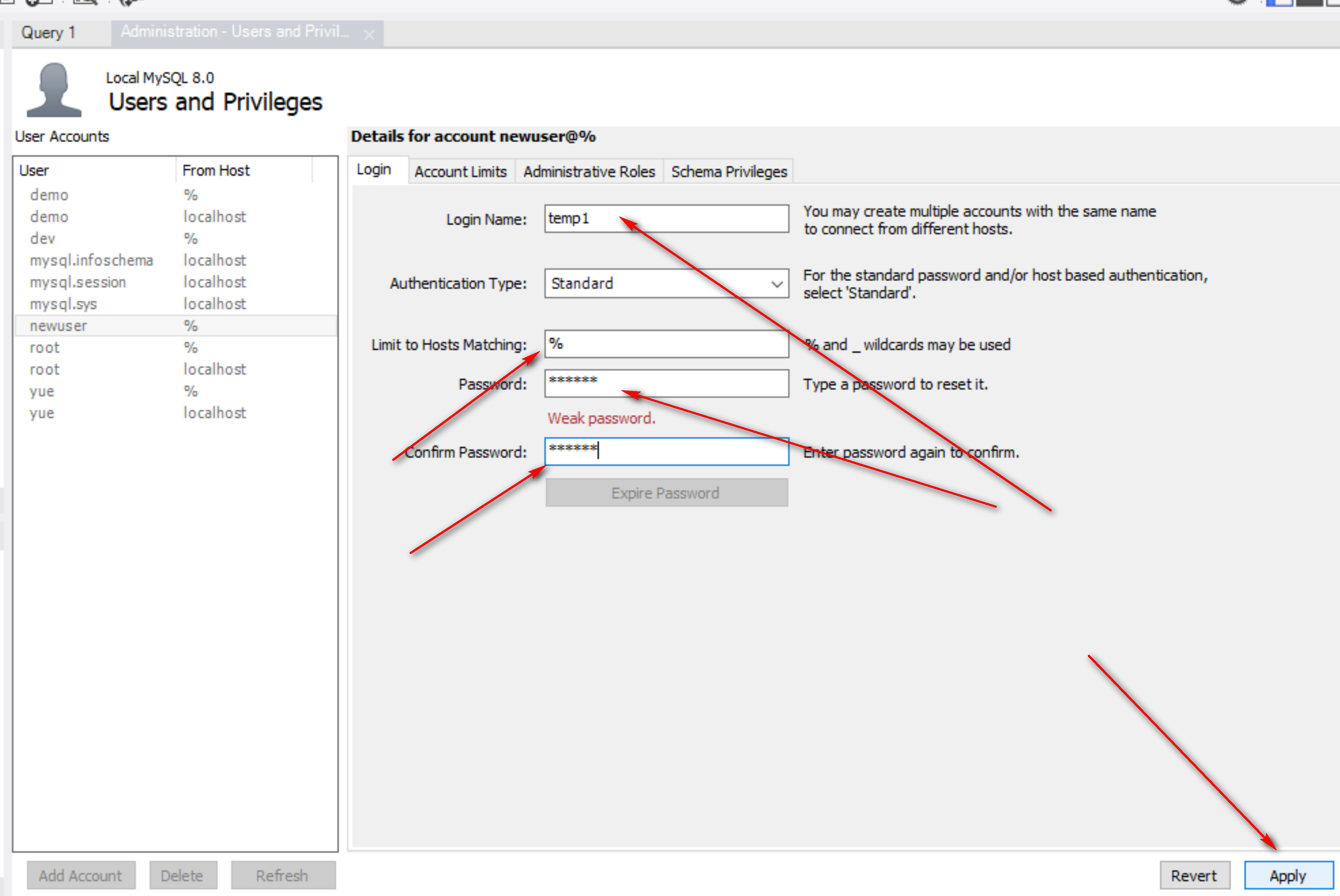


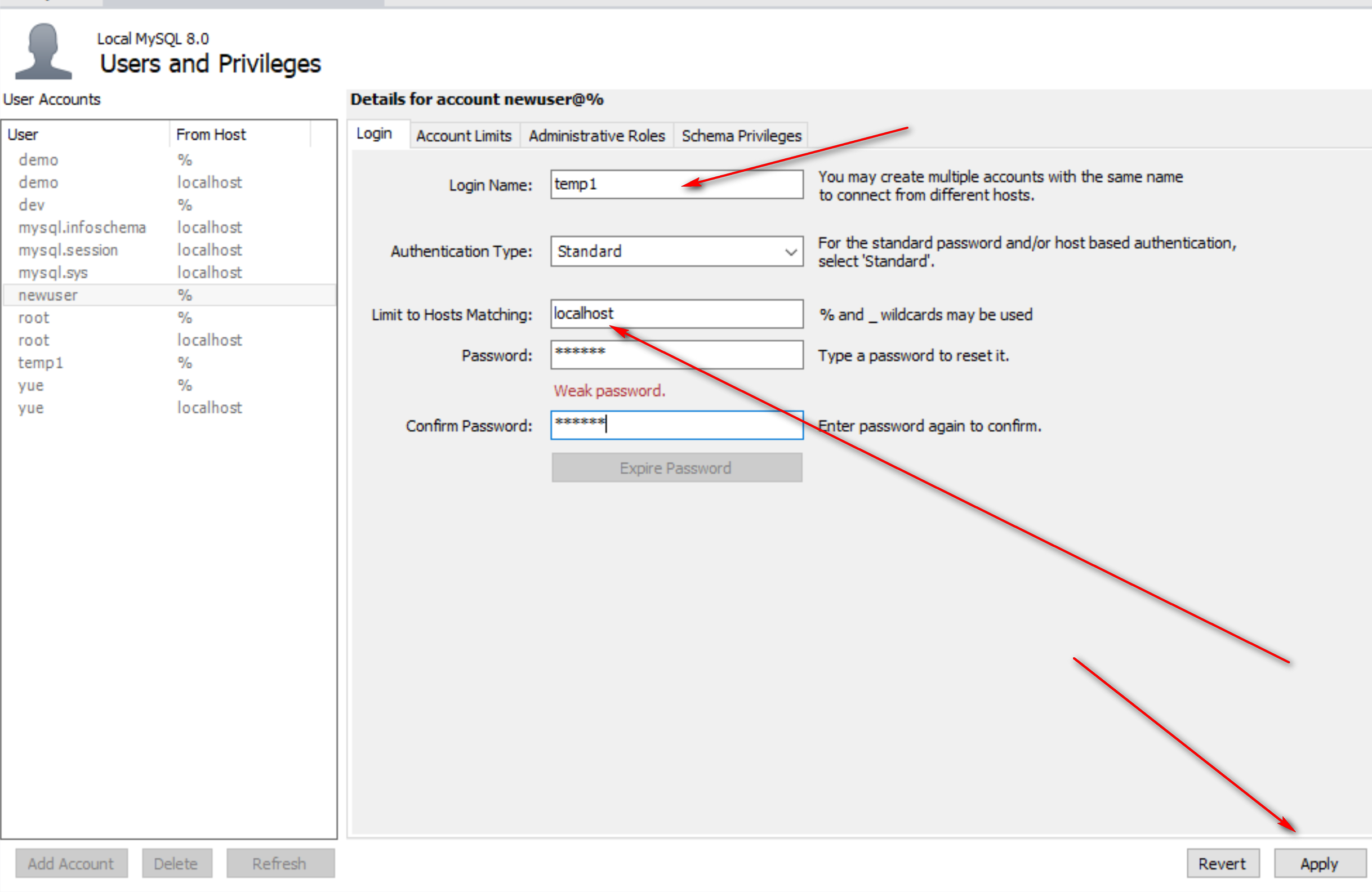


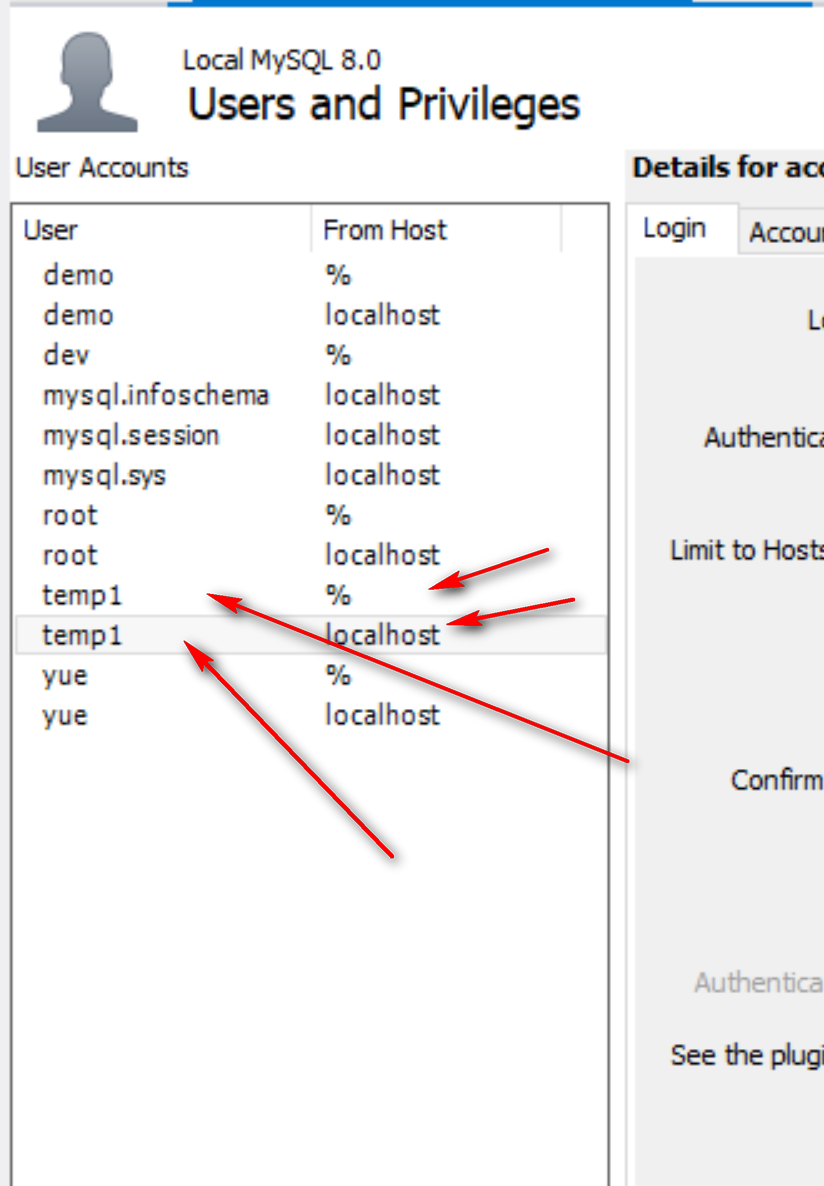


Account = user + host

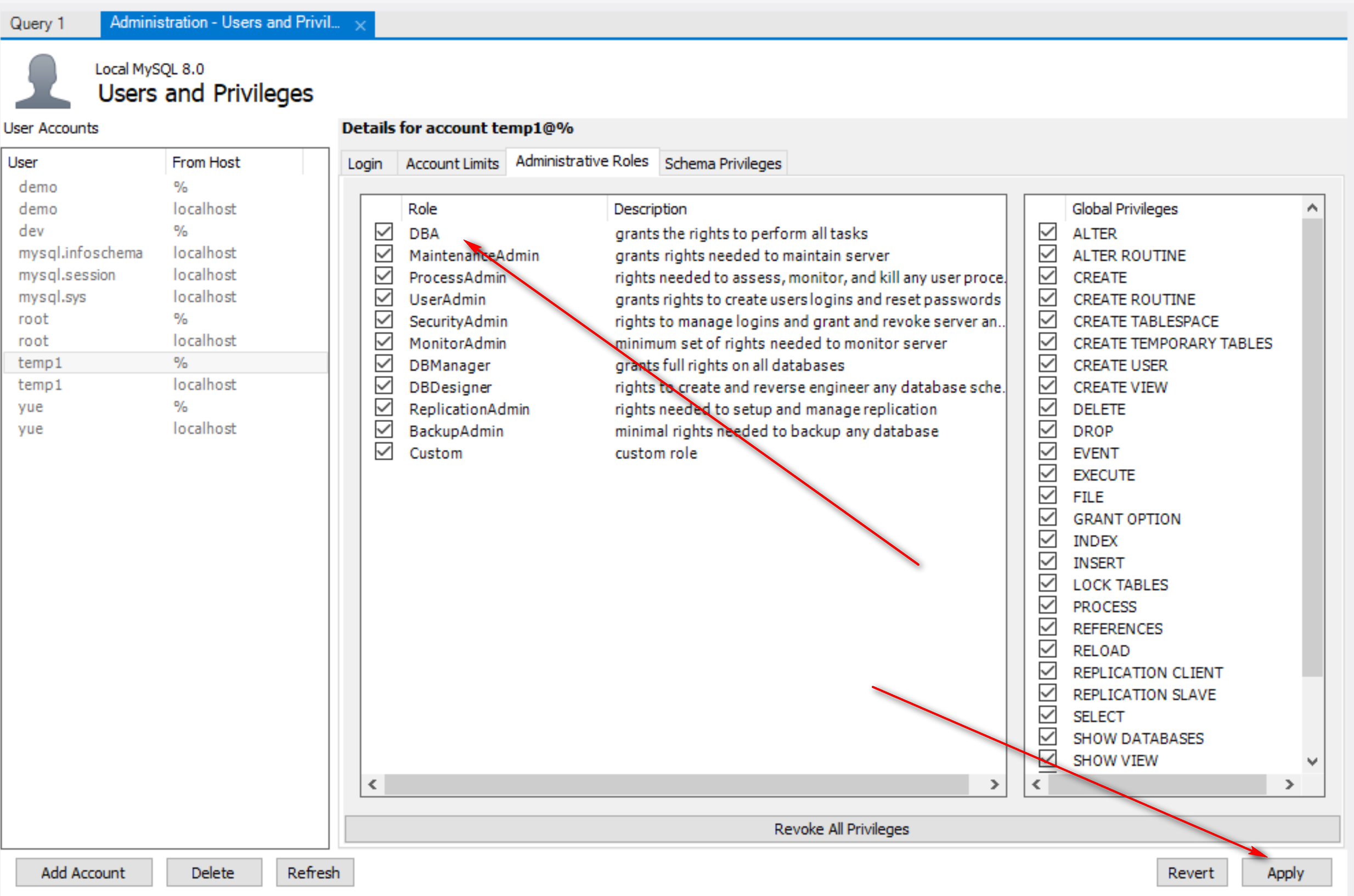
Host: % and localhost







Apply privileges:



* Embedded SQL: embedding SQL in a host language.
* SQL: DB language, not a general purpose language.
* Host languages:
  1. general purpose language
  2. usually have a vast library and tools
  3. may have a good community in a specific application area
* Examples of host languages: Java, C#, C, C++, PHP, Perl, Python, Ruby, etc.
* Preparation: create a MySQL account in your computer for development if you have not already done so. (You may use MySQL Workbench).
* Basic steps of embedded SQL programming.
  1. Making connections to the database
  2. Preparing and executing SQL statements through the connections
  3. Using the results of SQL statements
  4. House cleaning and closing DB connections
* Step (2) => learn your SQL well.
* Some issues in step (3):
  1. Data type mismatch: relations/tables usually not supported natively in the host languages.
  2. Performance consideration: DB operations are usually much more expensive (take longer time) than CPU operations.
  3. Security consideration: DB breaches are serious.
* To resolve the data type mismatch problem, the concept of *cursor* is usually used.
* A cursor allows the programmer to *iterate* through the result set, one row at a time.
* Cursor:
  1. PyMySQL: <http://pymysql.readthedocs.io/en/latest/modules/cursors.html>
  2. MySQL Connector: <http://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html>
* There may be different cursor types for performance, security, and access method consideration.

***Example:***

student\_1.py:

# Import pymysql connector to MySQL  
import pymysql  
  
# [1] Making connection to the MySQL server  
cnx = pymysql.connect(user='....', password='.....',  
    host='localhost',  
    database='toyu')  
  
# Create a cursor using the connection.  
cursor = cnx.cursor()  
  
# [2] Prepare a SQL query for the problem  
query = '''  
SELECT CONCAT (s.fname, ' ', s.lname) AS student,  
    d.deptName,  
    CONCAT(f.fname, ' ', f.lname) as advisor  
FROM student AS s LEFT JOIN department AS d  
        ON (s.major = d.deptCode)  
     LEFT JOIN faculty AS f  
        ON (s.advisor = f.facId);  
'''  
  
# Execute the query  
cursor.execute(query)  
  
# [3] Use the result in the query  
for (student, major, advisor) in cursor:  
    print("{}: major={}; advisor={}".format(student, major, advisor))  
  
# [4] Housekeeping  
cursor.close()  
cnx.close()

The output of running this program:

Tony Hawk: major=Computer Science; advisor=Paul Smith  
Mary Hawk: major=Computer Science; advisor=Paul Smith  
David Hawk: major=Computer Science; advisor=Paul Smith  
Catherine Lim: major=Information Technology; advisor=Deborah Gump  
Larry Johnson: major=Information Technology; advisor=Deborah Gump  
Linda Johnson: major=Computer Information Systems; advisor=Daniel Kim  
Lillian Johnson: major=Computer Information Systems; advisor=Daniel Kim  
Ben Zico: major=None; advisor=None  
Bill Ching: major=Arts; advisor=Art Allister  
Linda King: major=Arts; advisor=Art Allister

* Some tips:
  1. Test your SQL statements thoroughly first.
  2. Use typical input parameters for testing.
  3. Be mindful of:
     + security, especially SQL injection
     + special characters of the languages involved
     + performance
* It is better to use configuration module and data ini file (e.g. dbconfig.py and dbconfig.ini). Why?
  1. Reuse and maintenance
  2. Security
* In general, using a configuration data file is a good programming practice.

dbconfig.py:

import configparser  
  
#  simplistic and no error handling.  
def get\_mysql\_param(filename='dbconfig.ini', section='mysql'):  
  
    config = configparser.ConfigParser()  
    config.read(filename)  
  
    return config['mysql']

dbconfig.ini:

[mysql]  
host = localhost  
database = toyu  
user = your\_account  
password = your\_password

In your Python program:

from dbconfig import \*  
import pymysql  
  
db = get\_mysql\_param()  
cnx = pymysql.connect(user=db['user'], password=db['password'],  
                      host=db['host'],  
                      database=db['database'])  
cursor = cnx.cursor()

* See Format Specification Mini-Language: <https://docs.python.org/3.3/library/string.html#formatspec>
* Your SQL statement may use placeholder parameters: %s.
* Parametrized statements are preferred.

Example from <http://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor-execute.html>:

insert\_stmt = (  
  "INSERT INTO employees (emp\_no, first\_name, last\_name, hire\_date) "  
  "VALUES (%s, %s, %s, %s)"  
)  
data = (2, 'Jane', 'Doe', datetime.date(2012, 3, 23))  
cursor.execute(insert\_stmt, data)  
  
select\_stmt = "SELECT \* FROM employees WHERE emp\_no = %(emp\_no)s"  
cursor.execute(select\_stmt, {'emp\_no': 2})

***Example with toyu:***

Get the school code as the command line argument.

***student\_2.py***

import pymysql  
import sys  
  
#   Use the school code in command line arguments  
#   to list all students majoring in a department  
#   in the school.  
# [1] Making connection to the MySQL server  
cnx = pymysql.connect(user='....', password='.....',  
    host='localhost',  
    database='toyu')  
  
# Create a cursor using the connection.  
cursor = cnx.cursor()  
  
# [2] Prepare a SQL query for the problem  
query = '''  
SELECT CONCAT (s.fname, ' ', s.lname) AS student,  
    d.deptName,  
    CONCAT(f.fname, ' ', f.lname) as advisor  
FROM student AS s LEFT JOIN department AS d  
        ON (s.major = d.deptCode)  
     LEFT JOIN faculty AS f  
        ON (s.advisor = f.facId)  
WHERE d.schoolCode = %s;  
'''  
  
# [2b] Get input values  
school\_code = sys.argv[1]  
  
# Execute the query  
cursor.execute(query, (school\_code,))  
  
# [3] Use the result in the query  
for (student, major, advisor) in cursor:  
    print("{}: major={}; advisor={}".format(student, major, advisor))  
  
# [4] Housekeeping  
cursor.close()  
cnx.close()  
  
  
Notes:

* %s is used as a placeholder.
* The second argument of cursor.execute is (school\_code',). Note the ',' following school\_code.
* (school\_code,) is not the same as (school\_code):
  1. (school\_code) is the same as school\_code: () is used to enforce precedence.
  2. (school\_code,) is a tuple: () is a tuple builder.