**11/25/2019**

*-- count the number of majors in CSCI*

**SELECT** **DISTINCT** \*

**FROM** student

**WHERE** major = 'CSCI';

**SELECT** **DISTINCT** **COUNT**(stuId) **AS** `Number of majors in CSCI`

**FROM** student

**WHERE** major = 'CSCI'

**GROUP** **BY** major;

**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>

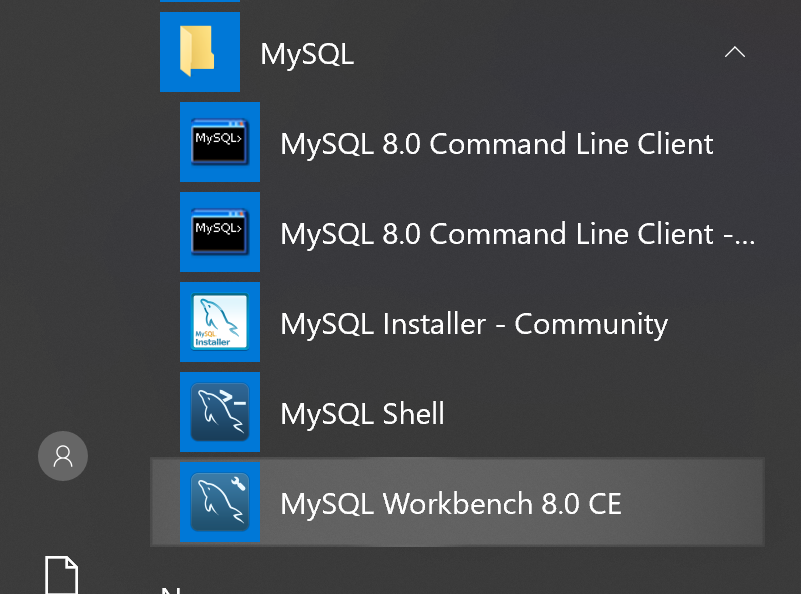
**Additional Materials: MySQL Connector Python (not used in this course, but it is a good resource)**

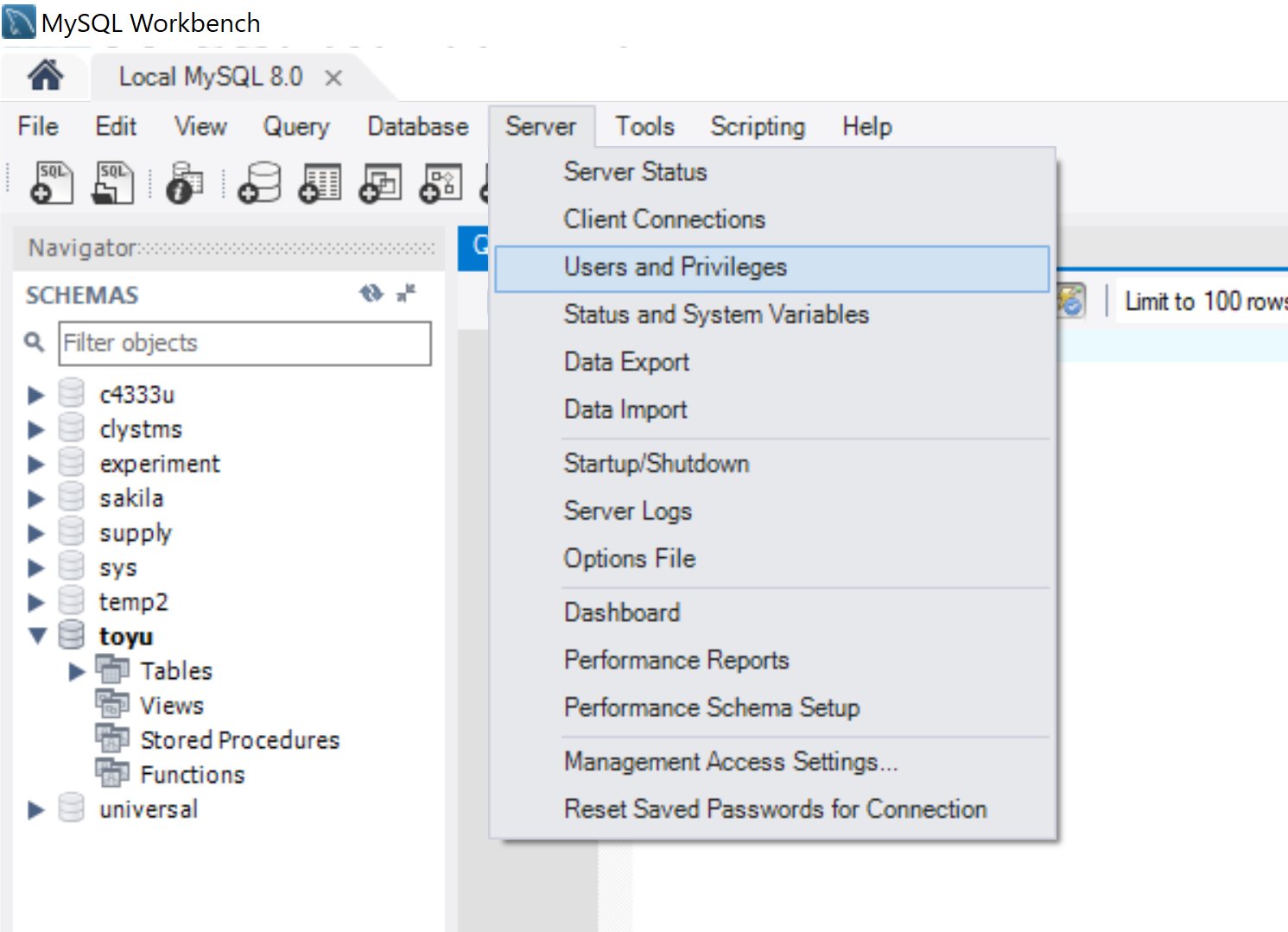
* API reference: <http://dev.mysql.com/doc/connector-python/en/connector-python-reference.html>
* Read MySQL Connector Python's Guideline for developer: [http://dev.mysql.com/doc/connector-python/en/connector-python-coding.htm](http://dev.mysql.com/doc/connector-python/en/connector-python-coding.html)
* 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>.

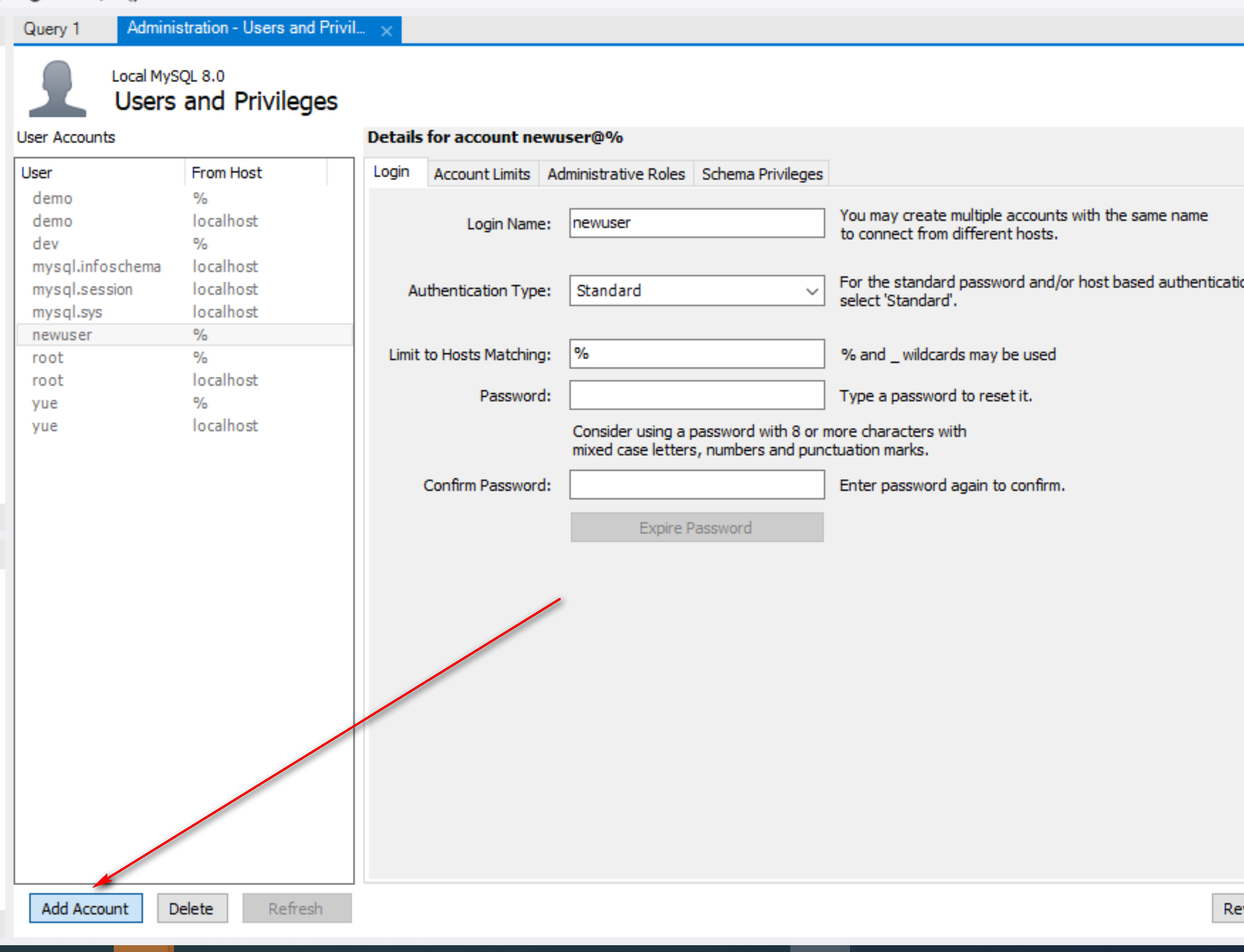
**Python Programming with MySQL**

Create a user account:

1/ Run MYSQL Workbench:

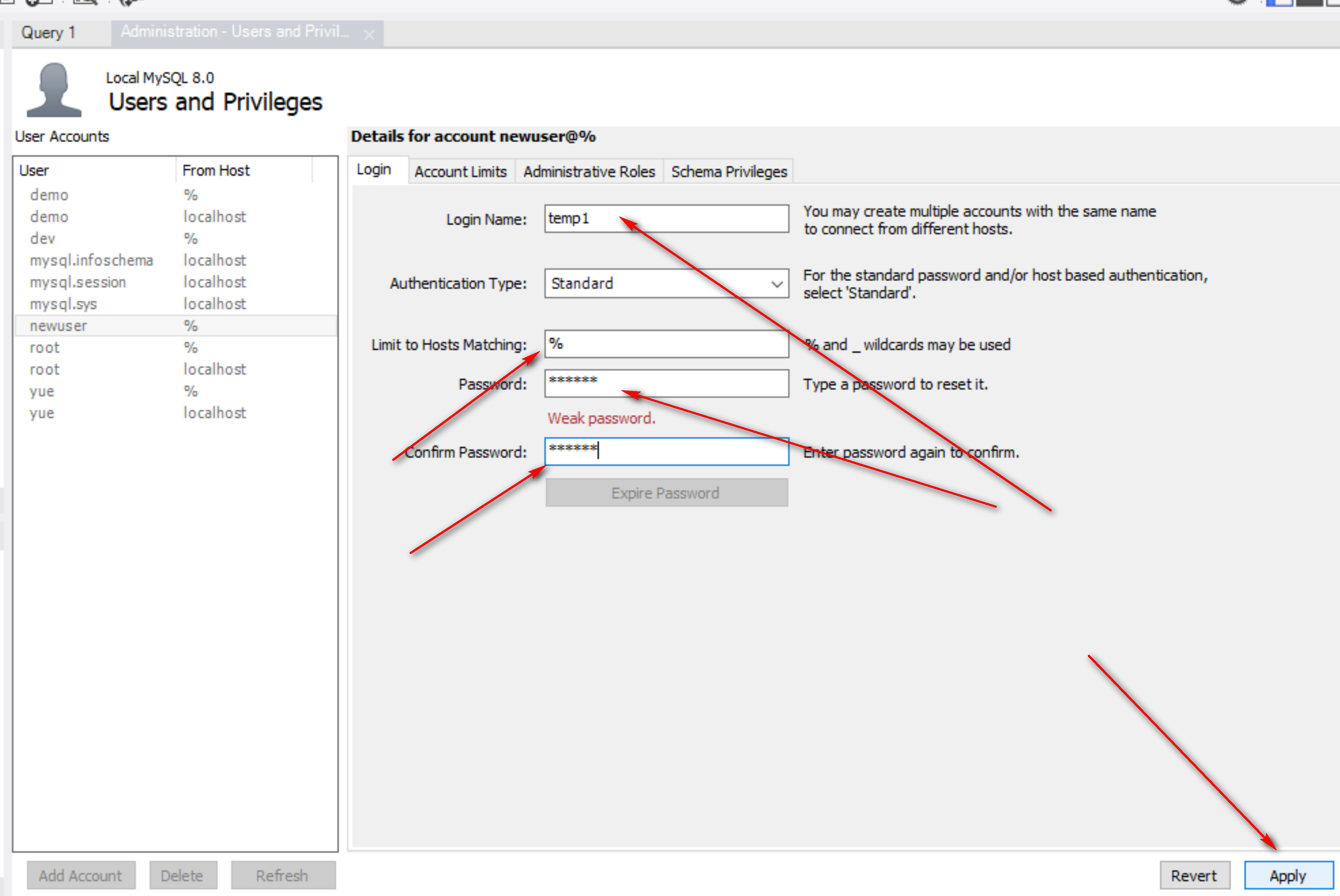


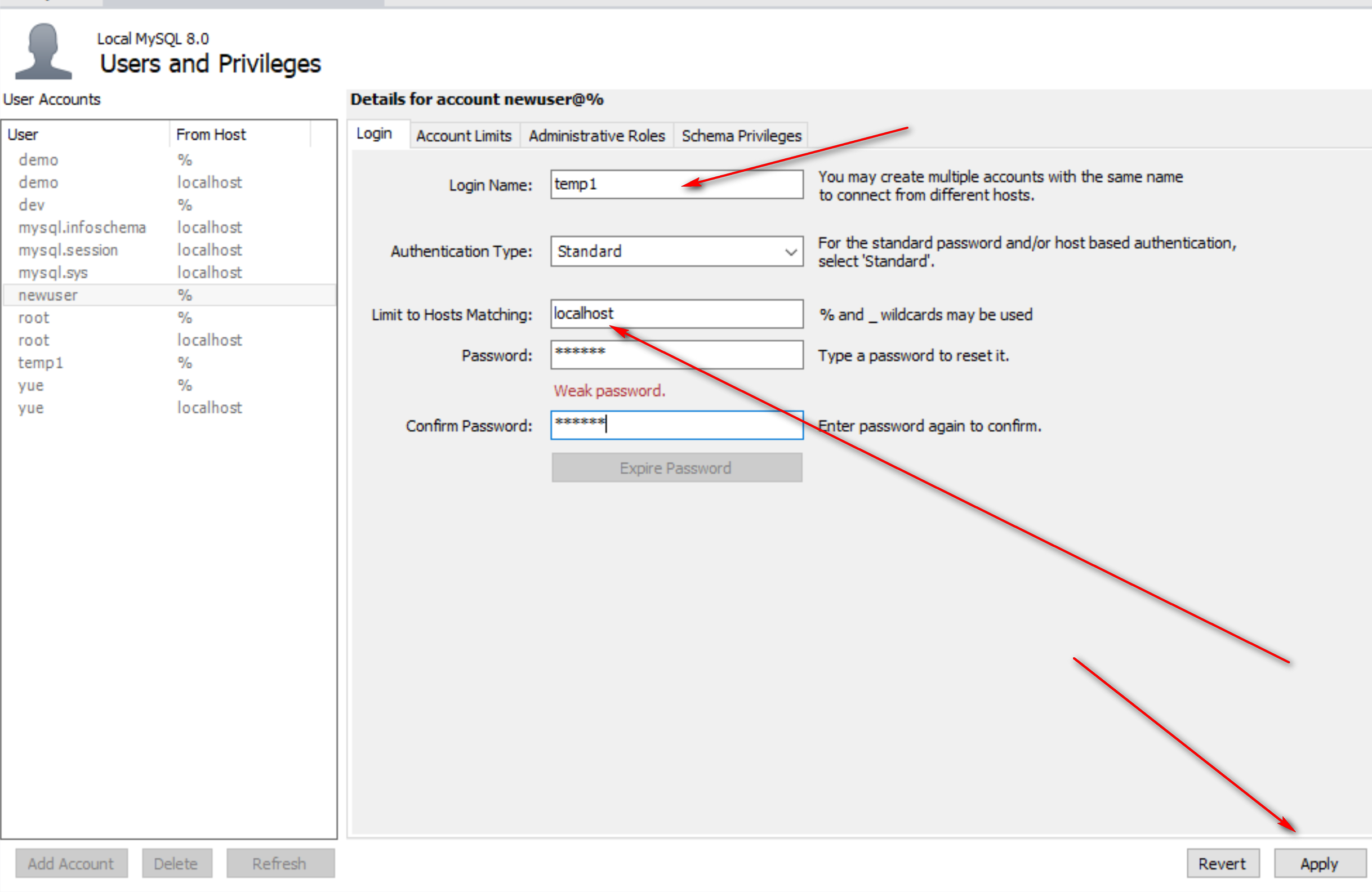


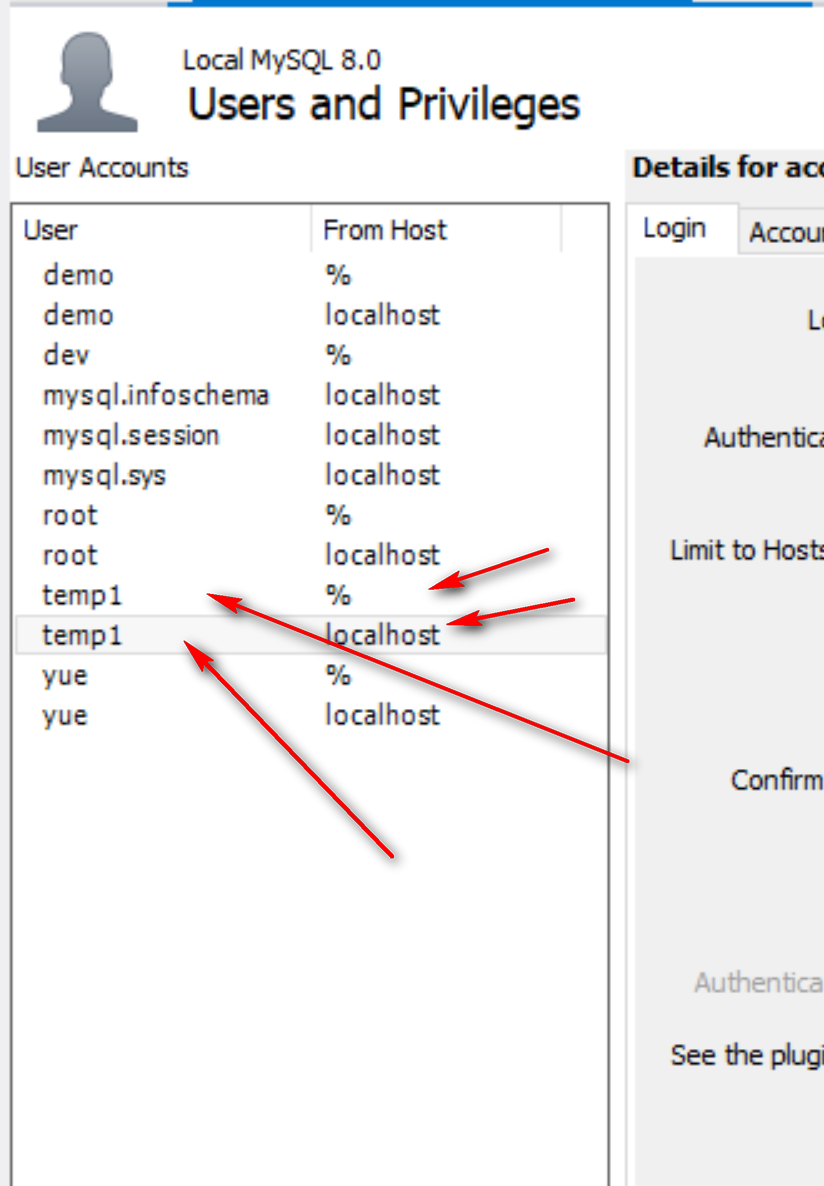


Account = user + host

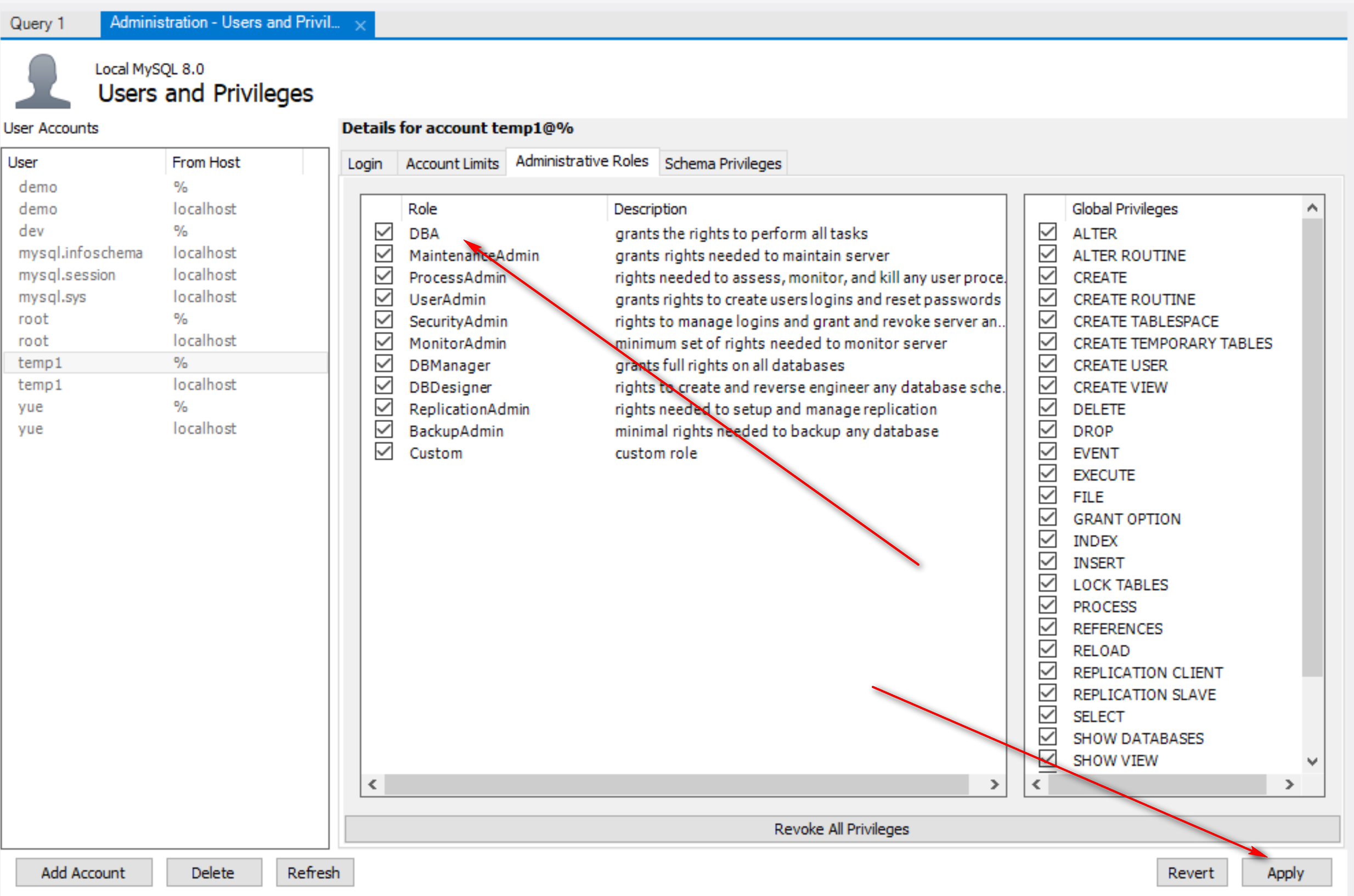
Host: % and localhost







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.