# CSCI 4333 Section 2 Design of DB Systems

## 4/1/2024 (self - annotation)

# 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 straightforward. Examples:
  + An example comparing driver performance: <https://gist.github.com/methane/90ec97dda7fa9c7c4ef1>
  + PyMySQL evaluation: <https://wiki.openstack.org/wiki/PyMySQL_evaluation>
* Because of some problems of MySQL Connector in Python 3, we use*PyMySQL.*

**2. PyMySQL:**

* Installation:
  + Install pip for Python, if necessary: <https://pip.pypa.io/en/stable/installing/>.
  + Command (may use: "pip install PyMySQL")
* Documentation: <http://pymysql.readthedocs.io/en/latest/>
* Compliant with Python Database API Specification v2.0: <https://www.python.org/dev/peps/pep-0249/#module-interface>.

|  |
| --- |
| **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." |

**3. Python Programming with MySQL**

* Embedded SQL: embedding SQL in a host language.
* SQL is a DB language, not a general purpose language.
* Host languages:
  1. general purpose language
  2. usually have vast libraries and tools
  3. usually have a good community in a specific application area
* Examples of popular host languages: Java, JavaScript, ts, GOLANG, 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 PHPMyAdmin or MySQL Workbench)

**3.1 Basic steps**

* Basic steps of embedded SQL programming.
  1. Create connections to the database
  2. Prepare and execute SQL statements through the connections
  3. Use the results of SQL statements
  4. Perform house cleaning and close DB connections
* Step (2) => learn your SQL well.
* Some issues in step (3):
  1. Data type mismatch (impedance 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. Examples:
     1. Minimize DB connections, especially active ones.
     2. Optimize the numbers and efficiency of SQL statements.
  3. Security consideration: DB breaches are serious.

**3.2 Connection and Cursor Objects**

* To solve 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 reference of PyMySQL: <http://pymysql.readthedocs.io/en/latest/modules/cursors.html>
* There may be different cursor types for performance, security, and access method considerations.
* PyMySQL's cursor types:
  1. SSCursor: unbuffered cursor
  2. DictCursor: return a dictionary
  3. SDictCursor: an unbuffered cursor returning a dictionary

***Example:***

[student1.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/emSQL/student1.py.txt): (All .py and .ini files have an extra extension, .txt, which should be removed when downloading).

import pymysql  
  
# [1] Making connection to the MySQL server  
cnx = pymysql.connect(user='demo', password='abcdef',  
    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:

C:\...>python student1.py  
Tony Hawk: major=Computer Science; advisor=Paul Smith  
Mary Hawk: major=Computer Science; advisor=Paul Smith  
David Hawk: major=Computer Science; advisor=Mary Tran  
Catherine Lim: major=Information Technology; advisor=None  
Larry Johnson: major=Information Technology; advisor=Deborah Gump  
Linda Johnson: major=Computer Information Systems; advisor=Daniel Kim  
Lillian Johnson: major=Computer Information Systems; advisor=Andrew Byre  
Ben Zico: major=None; advisor=None  
Bill Ching: major=Arts; advisor=None  
Linda King: major=Arts; advisor=Art Allister  
Cathy Johanson: major=None; advisor=Art Allister

* Some tips:
  1. Test your SQL statements thoroughly first.
  2. Use typical input parameters for testing.
  3. Be mindful of:
     1. security, especially SQL injection
     2. special characters of the languages involved
     3. performance

**3.3 Configuration files**

* It is better to use configuration module and data ini file (e.g. dbconfig.py and dbconfig.ini). Why?
  1. Separation of data and program
  2. Reuse and maintenance
  3. Security
* In general, using a configuration data file is a good programming practice.

[dbconfig.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/emSQL/dbconfig.py.txt):

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

If running in Windows IIS (such as in the dcm server):

import configparser  
from pathlib import Path  
  
#  simplistic and no error handling.  
def get\_mysql\_param(filename='dbconfig.ini', section='mysql'):  
    config = configparser.ConfigParser()  
    file\_path = (Path(\_\_file\_\_).parent / filename).resolve()  
    config.read(file\_path)  
  
    return config[section]

[dbconfig.ini](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/emSQL/dbconfig.ini.txt):

[mysql]  
host = localhost  
port = 3306  
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'],  
      port=int(db['port']))  
cursor = cnx.cursor()

If you use the default port (i.e., 3306), you may adjust by

1. remove the line "port=3306" in dbconfig.ini, and
2. remove the argument "port" in the call of pymysql.connect() in the main program.

In your Python program,

[student2.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/emSQL/student2.py.txt):

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

* See Format Specification Mini-Language: <https://docs.python.org/3.11/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:***

Get the school code as the command line argument.

[student3.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/emSQL/student3.py.txt)

from dbconfig import \*  
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  
  
db = get\_mysql\_param()  
cnx = pymysql.connect(user=db['user'], password=db['password'],  
      host=db['host'],  
      database=db['database'],  
      port=int(db['port']))  
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 = 'CSE'  
if len(sys.argv) > 1:  
    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()  
  
Result:

C:\...>python student3.py CSE  
Linda Johnson: major=Computer Information Systems; advisor=Daniel Kim  
Lillian Johnson: major=Computer Information Systems; advisor=Andrew Byre  
Tony Hawk: major=Computer Science; advisor=Paul Smith  
Mary Hawk: major=Computer Science; advisor=Paul Smith  
David Hawk: major=Computer Science; advisor=Mary Tran  
Catherine Lim: major=Information Technology; advisor=None  
Larry Johnson: major=Information Technology; advisor=Deborah Gump  
   
Notes:

* %s is used as a placeholder.

Setup DCM web hosting and FTP;

A screenshot of a computer

Description automatically generated

**Web Database Development using MySQL/Python/CGI**

by K. Yue

**1. Python in Web Development**

* HOWTO Use Python in the web: <https://docs.python.org/3.3/howto/webservers.html>

**1.1 Set up Python on Apache in XAMPP with barebone CGI**

In the httpd.conf (Apache startup configuration file), which is likely in c:\xampp\apache:

1. Enable CGI by adding the line "LoadModule cgi\_module modules/mod\_cgi.so"
2. Find the line "AddHandler cgi-script" and add ".py" at the end. E.g. "AddHandler cgi-script .cgi .pl .asp .py"
3. Add the line "ScriptInterpreterSource Registry-Strict" at the end: this is telling Apache to use Windows registry to identify the interpreter.

This will set Apache to call Python interpreter to handle Web requests with the extension .py.

A major advantage of CGI in a simple course like ours is its simplicity. Well known Python Web platforms such as Django and Flask have many more features and complexity.

However, CGI in Python:

1. It is slow.
2. Furthermore, it is deprecated in Python version 3.11. We will select new mechanism in future semesters.

One common problem of Xampp\Apache\Python is that accessing your py page returns a 500+ web server error. One possibility is that Apache does not identify the right Python interpreter to interpret your Python page. In [3] ScriptInterpreterSource Registry, you inform Apache to use windows registry to identify the right Python interpreter and it may not be set correctly.

One solution is to add the *she-bang* command as the*very first* line of your Python page, which specifies the location of the Python interpreter. There should be no character before "#!". E.g.: in the first line of your Python Web program.

#!C:\python311\Python.exe

Test it with a Web page such as test.py (from HOWTO Use Python in the web). In test.py below, there are also code added to disable the program to send warnings to the Web server.

[test.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/web/test.py.txt)

#!"c:\python311\python.exe"  
import warnings  
warnings.filterwarnings('ignore')  
# enable debugging  
import cgitb  
cgitb.enable()  
  
print("Content-Type: text/html;charset=utf-8")  
print()  
print("<html><body>Hello World!</body></html>")

You may need to set the shebang line to point to your Python interpreter. Your version may be more recent and your path may be different.

**1.2 Python-CGI**

* Common Gateway Interface (CGI) is a specification standard on how a Web server executes an application program.
* There are many ways to develop Python's Web applications.
* In this course, we focus on the most fundamental (and primitive) one, Python CGI: <https://docs.python.org/3.5/library/cgi.html#using-the-cgi-module>.
* Realistic Web application development should use a more sophisticated platform such as Django.

In the example above,

1. print("Content-Type: text/html;charset=utf-8") will print to the standard output, such as the command line prompt.
2. Under CGI, when the Web server invokes a Python program, output to the standard output will send the output to the Web Server.
3. Note that there are many newer standards and methods for a web server to execute a program.

**2. Static Web Pages**

**2.1 Steps in static Web page development**

1. A user agent makes a request to the Web server with an URL: HTTP Request to the Web server in the URL.
2. The Web server gets the requested resource:
   1. The URL is mapped by the Web server to a local resource:*URL Mapping* maps URL to a *data* file, usually HTML.
   2. The Web server fetches (reads) the file.
3. The Web server prepares a HTTP Response, the body of which is the fetched file.

A diagram of a web page

Description automatically generated

If localhost is used in the HTTP Request, the server computer is also the user device.

URL: <http://localhost/python/joindb/S2024/s2/index.html>

Mapped to:

C:\xampp\htdocs\python\joindb\S2024\s2\index.html

<http://localhost/python/joindb/S2024/s2/test.py>

->

C:\xampp\htdocs\python\joindb\S2024\s2\test.py

A diagram of a server

Description automatically generated

DCM:

<https://dcm.uhcl.edu/c433324sp02guevaraa/>demo/s2/index.html ->

DCM file: c433324sp02guevaraa/pages/demo/s2/index.html

<https://dcm.uhcl.edu/yue>/demo/s2/test.py ->

->

yue/pages/demo/s2/test.py

**2.2 HTTP**

***HTTP Request:***

1. HTTP Request header: storing meta data in name-value pairs
2. An empty line
3. HTTP Request body (optional)

***HTTP Response***

1. HTTP Response header in name-value pairs.
2. An empty line
3. HTTP Response body (optional)

There are many tools that you can use to look at HTTP request and response headers. For example, for Chrome, One may be [using the built-in inspector](https://www.mkyong.com/computer-tips/how-to-view-http-headers-in-google-chrome/) (Control -> Inspect -> Network).

***Example:***

Using Chrome to access http://dcm.uhcl.edu/yue/:

HTTP Request header may be:

GET /yue/ HTTP/1.1  
Host: dcm.uhcl.edu  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8  
Accept-Encoding: gzip, deflate, sdch  
Accept-Language: en-US,en;q=0.8  
Cookie: ASPSESSIONIDSQCQRSBD=INENHEEBLFGOGHKFBKIBLAEO; ASPSESSIONIDSQBRRTAC=BHCPCNOBKEBCIGBIAPPGKCCC; ASPSESSIONIDSSCTRSAD=MICBCLJCAEMDBIBIBNBGGIIE; ASPSESSIONIDSSCRSSAC=GKGLGNDDFIMKDKBHLKPDPCHO; ASPSESSIONIDSQBTTRAC=CBABGKODOJHMDCNONKOCPDHN; ASPSESSIONIDSQARTSBC=IGNDLEDDOJEGAGCMBLEICEEI; ASPSESSIONIDQSARTSAD=PKLPPBDBNCHJPBLALOPCJIGN; ASPSESSIONIDSSARRRDD=CMPNGIHAKBPNAOKKGOGLHPFK; ASPSESSIONIDQSARSTAD=PCADGENBBDKFJEMAMDIKEHOB; ASPSESSIONIDQSDTTQBD=JKHNECHCBMCLEHFBGHFMFELC; ASPSESSIONIDSSARRRCC=BAJNJNGAPJOOBMIDAPOIHJHN; ASPSESSIONIDACSSCTQR=CAJBMBBDMCGDNOLIJBGHEEIO; ASPSESSIONIDCARSAQTQ=KFCHGJABODHAJJCJBPNKMICK; ASPSESSIONIDCQRACTCB=POPHJGADHNGFECOGGEOIOFBA; ASPSESSIONIDCQTCARBD=FOBNDOABCKNJBGPMMFFCHFJJ; ASPSESSIONIDASTCARBC=KDHLNLKBDLAPGEIMILDNOFCB; ASPSESSIONIDCQSDCTCB=HPOBPEECPBNPPBIFPMKPHKAL; ASPSESSIONIDCSQTARAS=BDHFPJDCNCAEPFFMFPABAMKM; ASPSESSIONIDQSSSRRTA=CCPDFCHBILLMAFIMLLNBPOAB; ASPSESSIONIDSQSRRQTA=CMDFCOGDGBHDFAHIIADBABJO; ASPSESSIONIDQSSQSQSA=LJOBGGBAEBNGBODOMKDPFLCF; ASPSESSIONIDQSSRQTTB=GFHPPMMAJMDOCAOBKNKHFLPP; ASPSESSIONIDSSQRTRCS=HKOBDDLAABMCJPGNIGFNKMIJ; ASPSESSIONIDSSTTSTBT=DMMLKGKANJPNILMPOAPMHLID; ASPSESSIONIDCARTQCAS=MJMFMKEDNCKKINPHLCIOGNPA; ASPSESSIONIDAATQTDBS=GPGPOCPDJGFLGEGKGHCLJABE; ASPSESSIONIDACRSQCAT=EDPJEGEBEGAHOFNKLLJCLKBJ; ASPSESSIONIDAAAATTBA=OFJJHHJCONLIOFGCCIHACKCG; ASPSESSIONIDAACBSSAA=BEMNHODDCMIHGABJKDONDLBC; ASPSESSIONIDAADASTBB=ECONFFOBCGANIFAOBCPJDIDH; ASPSESSIONIDAADCRSAA=NEIBBLICBGHFINHGPPFCJAKN; ASPSESSIONIDCCDBTRCA=OKFJCGDDMALOAABLJCKMHBBM; ASPSESSIONIDQQRTSRCQ=IKBJHDODEOFFFANBAHFKAAPI; ASPSESSIONIDSSSTQSDR=CGIJNOHCDGBILOOMDOFGLACH; ASPSESSIONIDQQSRTTDQ=IGLDFICDELFIENPMDINFCLJC; ASPSESSIONIDQCDDTTAQ=IPDJJHNDIHKHJLLPOMHGGLCF; ASPSESSIONIDQCDCRQDQ=ONCNOFBBKPGLDEIGIALKHKPO; ASPSESSIONIDQCAAQSDQ=LEHFKPLDJNMEDOCIHBKDMMEH; ASPSESSIONIDQAADRRDQ=LMNNECGAPNGBLONGIMFFOEAB; ASPSESSIONIDACRCRTQD=BPOHFIKBGEBMEEGOBKKBJAGD; ASPSESSIONIDAATDSRRD=LMGJCJPCJLCBJOOIPLFLNCAA; ASPSESSIONIDSQACSRDR=DGNJNLJBLOMAHDEAAOEFCAMM; ASPSESSIONIDQQADQSDQ=PJJFHIOAAGKDJDAFKDIGOMJN; ASPSESSIONIDQSCDQTCQ=JKCHLINCGEHJDFEGHNIBKMIM; ASPSESSIONIDSSBCSSDR=MCGPECIDJNGIEFBMLEHKNOKJ; ASPSESSIONIDSSDCQTDR=COIDIPBAMMBLPGFKCKDEPMNG; ASPSESSIONIDSACSBBTR=DIKJFGMAEDOKGKBFFGANFLNF; \_\_utmc=61056616; ASPSESSIONIDCATSDDST=HIBNGHNDHJLCEKFOFAEDHNPA; ASPSESSIONIDAAQSCDTS=FADLJFICPIFBOGEPFAPEGNIE; ASPSESSIONIDASQRTTAQ=FPHHDCPAGFFKAKCCKEABAJBJ; ASPSESSIONIDAQRTQQDQ=PCPHCJDCOCCGMJOIMDBOKGHG; ASPSESSIONIDASQSTSAQ=KHFDEONCEBMMCACPPGDMNKDB; has\_js=1; ASPSESSIONIDQAADDRQA=DPCFGJLALPHKCEFJLCMAIMNJ; ASPSESSIONIDSCCQASAR=CLKFPHBDNCPCDAPGFHOBJEMA; ASPSESSIONIDQAATATBR=NNEHOJMDJCFNKGNFPMDHPJNM; ASPSESSIONIDSQTTCDTA=HKJDFDAABHHHOPGFFDOHDFDO; ASPSESSIONIDSSSQADSC=DDNBIEOBDFJCLODCMNFHINHA; \_\_utma=66415163.264850463.1309620985.1410922342.1428450784.123; \_\_utmc=66415163; \_\_utmz=66415163.1428450784.123.1.utmccn=(direct)|utmcsr=(direct)|utmcmd=(none); .UHCLAuthNet=89A9D736D3B96923E9BCB6E6C53E80E0DDC8DDD4BD0D724749FD14CFF04168C3FBBCDEF2182008646D15199DA8CDF39711261C2BC52AA1EE77D35E00B6FF5C66B3C0BBEE170AC1AB582A889BAA0419EFB36052C406E4372450A65406A8D11844E4828C4802058711568C15831E272DE827ED0C3E; WT\_FPC=id=172.29.1.35-3959882976.30274169:lv=1438888300784:ss=1438888300784; \_ga=GA1.2.903118926.1438958881  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.155 Safari/537.36

HTTP Response header:

HTTP/1.1 200 OK  
Accept-Ranges: bytes  
Content-Encoding: gzip  
Content-Length: 1970  
Content-Type: text/html  
Date: Wed, 12 Aug 2015 19:50:09 GMT  
ETag: "3be2b1c6662fd01:0"  
Last-Modified: Tue, 13 Jan 2015 19:26:06 GMT  
Server: Microsoft-IIS/7.5  
Vary: Accept-Encoding  
X-Powered-By: ASP.NET

For [test.html](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/web/test.html):

If you have curl (<https://curl.se/>) in your computer, you may use

curl -i localhost/python/joindb/test.html

(replace with your favorite url) to get something like:

curl -i localhost/python/test.html  
HTTP/1.1 200 OK  
Date: Fri, 23 Dec 2022 22:16:59 GMT  
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.5  
Last-Modified: Fri, 23 Dec 2022 22:16:53 GMT  
ETag: "2f-5f086249824d1"  
Accept-Ranges: bytes  
Content-Length: 47  
Content-Type: text/html  
  
<html>  
<body>  
Hello, world.  
</body>

**3. Dynamic Web Pages**

**3.1 Dynamic Web page generation using CGI**

1. A user agent makes a request to the Web server with an URL: HTTP Request to the Web server in the URL.
2. The Web Server invokes the requested resource program:
   1. The URL is mapped by the Web server to a local resource: URL Mapping maps URL to a *program* file, such as .aspx, .php, .cgi, .pl, .py, etc.
   2. The Web server invokes the appropriate interpreter to *execute* the program.
   3. Communications between the Web server and the program use the *CGI* standard.
3. The program executes:
   1. It can interact with other software and resources.
   2. Standard output from the program is sent to the Web server.
4. The Web server prepares a HTTP Response accordingly.

A diagram of a server system

Description automatically generated

***Example:***

In the program [test.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/web/test.py.txt)

#!"c:\python310\python.exe"  
# enable debugging  
import cgitb  
cgitb.enable()  
  
print("Content-Type: text/html;charset=utf-8")  
print()  
print("<html><body>Hello World!</body></html>")

the Python CGI program output:

1. A part of the HTTP Response header: "Content-Type: text/plain;charset=utf-8"
2. An empty line
3. THe HTTP Response body: "Hello World"

The Web Server will 'dress up' the HTTP Response Header.

If you have curl (<https://curl.se/>) in your computer, you may use

curl -i localhost/python/joindb/s2023/test.py

(replace with your favorite url) to get something like:

curl -i localhost/python/test.html  
HTTP/1.1 200 OK  
Date: Fri, 23 Dec 2022 22:16:59 GMT  
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.5  
Last-Modified: Fri, 23 Dec 2022 22:16:53 GMT  
ETag: "2f-5f086249824d1"  
Accept-Ranges: bytes  
Content-Length: 47  
Content-Type: text/html  
  
<html>  
<body>  
Hello, world.  
</body>

* The Python program is used to implement the logic of the Web application, including database access.

**3.2 Common Gateway Interface (CGI)**

* From Wikipedia: "Common Gateway Interface (CGI) is an interface specification that enables web servers to execute an external program, typically to process user requests."

***Example***

[csciStudent1.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/web/csciStudent1.py.txt):

#!"c:\python310\python.exe"  
from dbconfig import \*  
import pymysql  
import warnings  
warnings.filterwarnings('ignore')  
# cgi: Support module for Common Gateway Interface (CGI) scripts.  
# cgitb: Traceback manager for CGI scripts  
# cgitb.enable(): enable trace back feature  
import cgi  
import cgitb  
cgitb.enable()  
  
#   Establish a cursor for MySQL connection.  
db = get\_mysql\_param()  
cnx = pymysql.connect(user=db['user'],  
                      password=db['password'],  
                      host=db['host'],  
                      # port needed only if it is not the default number, 3306.  
                      # port = int(db['port']),  
                      database=db['database'])  
                               
cursor = cnx.cursor()  
  
#   Create HTTP response header  
print("Content-Type: text/html;charset=utf-8")  
print()  
  
#   Create a primitive HTML starter  
print ('''<html>  
<head></head>  
<body>  
''')  
  
  
query = '''  
SELECT DISTINCT s.stuId,  
    CONCAT(s.fname, ' ', s.lname) AS student,  
    s.ach,  
    IFNULL(CONCAT(f.fname, ' ', f.lname), 'N/A') AS advisor  
FROM toyu.student AS s LEFT JOIN toyu.faculty AS f ON (s.advisor = f.facId)  
WHERE s.major = 'CSCI'     
'''  
  
cursor.execute(query)  
  
# Read data and generate code for a HTML table.  
print('''  
<table border='1'>  
<tr><th>Student Id</th><th>Name</th><th>Accumulated credits</th><th>advisor</th></tr>  
''')  
  
print("<h3>CSCI student info</h3>")  
for (stuId, student, credits, advisor) in cursor:  
    print("<tr><td>{}</td><td>{}</td><td>{}</td><td>{}</td></tr>".format(stuId, student, credits, advisor))  
  
cursor.close()  
cnx.close()  
  
print ('''  
</table>  
</body>  
</html>  
''')  
  
The Python program sends the following output to the Web server:

Content-Type: text/html;charset=utf-8  
  
<html>  
<head></head>  
<body>  
  
  
<table border='1'>  
<tr><th>Student Id</th><th>Name</th><th>Accumulated credits</th><th>advisor</th></tr>  
  
<h3>CSCI student info</h3>  
<tr><td>100000</td><td>Tony Hawk</td><td>40</td><td>Paul Smith</td></tr>  
<tr><td>100001</td><td>Mary Hawk</td><td>35</td><td>Paul Smith</td></tr>  
<tr><td>100002</td><td>David Hawk</td><td>66</td><td>Mary Tran</td></tr>  
  
</table>  
</body>  
</html>

The HTTP response by the Web server:

curl -i http://localhost/.../csciStudent1.py

result:

HTTP/1.1 200 OK  
Date: Fri, 23 Dec 2022 23:02:35 GMT  
Server: Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.5  
Transfer-Encoding: chunked  
Content-Type: text/html;charset=utf-8  
  
<html>  
<head></head>  
<body>  
  
  
<table border='1'>  
<tr><th>Student Id</th><th>Name</th><th>Accumulated credits</th><th>advisor</th></tr>  
  
<h3>CSCI student info</h3>  
<tr><td>100000</td><td>Tony Hawk</td><td>40</td><td>Paul Smith</td></tr>  
<tr><td>100001</td><td>Mary Hawk</td><td>35</td><td>Paul Smith</td></tr>  
<tr><td>100002</td><td>David Hawk</td><td>66</td><td>Mary Tran</td></tr>  
  
</table>  
</body>  
</html>  
  
Output may look like:

A screenshot of a computer

Description automatically generated

If you want to use dbconfig.py and dbconfig.ini in the dcm server (running IIS), you may need to use the following code that works for Windows IIS.

dbconfig.py:

import configparser  
from pathlib import Path  
  
#  simplistic and no error handling.  
def get\_mysql\_param(filename='dbconfig.ini', section='mysql'):  
    config = configparser.ConfigParser()  
    file\_path = (Path(\_\_file\_\_).parent / filename).resolve()  
    config.read(file\_path)  
    return config[section]

This is needed because the relative filename in Python script in IIS Web server in DCM is not relative to the script directory.

dbconfig.ini for HW assignment hosted in the DSM server:

Method #1:

[mysql]  
host = localhost  
database = swim  
user = <your dcm MySQL user account: e.g., yueb>  
password = <your dcm MySQL account password: e.g., Sce1234567!!>

Method #2:

Use the predefined account in DCM: will be discussed in the class.

[mysql]  
host = localhost  
database = swim  
user = dbguest  
password = <<to be disclosed in the class>>

* HTTP provides various methods for the user agents to submit data to the Web servers.
* Data are submitted as name-value paired *parameters*.
* The main ones are GET and POST:
* GET:
  1. The parameters are a part of the URL after '?', known as the *query string*.
  2. The HTTP request body is empty.
* POST:
  1. The parameters are not a part of the URL.
  2. The HTTP request body stores the parameters.
* In Python CGI module, no matter whether GET or POST are used, HTTP parameters can be obtained through the dictionary cgi.FieldStorage().

***Example:***

[cgi1.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/web/cgi1.py.txt):

#!"c:\python310\python.exe"  
import cgi  
import cgitb  
cgitb.enable()  
  
print("Content-Type: text/html;charset=utf-8")  
print()  
  
print ('''<html>  
<head></head>  
<body>  
''')  
  
form = cgi.FieldStorage()  
if "name" not in form or "addr" not in form:  
    print("<H1>Error</H1>")  
    print("Please fill in the name and addr fields.")  
    quit()  
print("<p>name:", form["name"].value)  
print("<p>addr:", form["addr"].value)  
  
print ('''</body>  
</html>  
''')

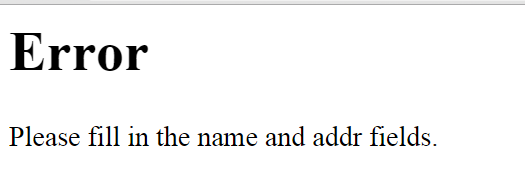
For: http://localhost/.../cgi1.py

or

http://localhost/.../cgi1.py?name=Yue

or

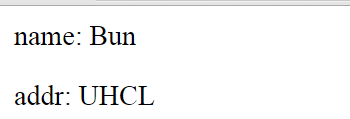
http://localhost/.../cgi1.py?Addr=UHCL



The data sent by the program to the Web server:

Content-Type: text/html;charset=utf-8  
  
<html>  
<head></head>  
<body>  
  
<H1>Error</H1>  
Please fill in the name and addr fields.

For: http://localhost/.../cgi1.py?name=Bun&addr=UHCL:



The data sent by the program to the Web server:

Content-Type: text/html;charset=utf-8  
  
<html>  
<head></head>  
<body>  
  
<p>name: Yue  
<p>addr: UHCL  
</body>  
</html>

The actual HTTP Response sent by the Web Server may be:

HTTP/1.1 200 OK  
Connection: Keep-Alive  
Content-Type: text/html;charset=utf-8  
Date: Thu, 13 Aug 2015 15:45:23 GMT  
Keep-Alive: timeout=5, max=100  
Server: Apache/2.2.21 (Win32) mod\_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod\_perl/2.0.4 Perl/v5.10.1  
Transfer-Encoding: chunked  
  
<html>  
<head></head>  
<body>  
  
<p>name: Yue  
<p>addr: UHCL  
</body>  
</html>

* FieldStorage provides other useful utilities. For example:
  + getfirst(name[, default]): get the first value of a *multi-valued*parameter.
  + getlist(name): get the entire list of values of a parameter.

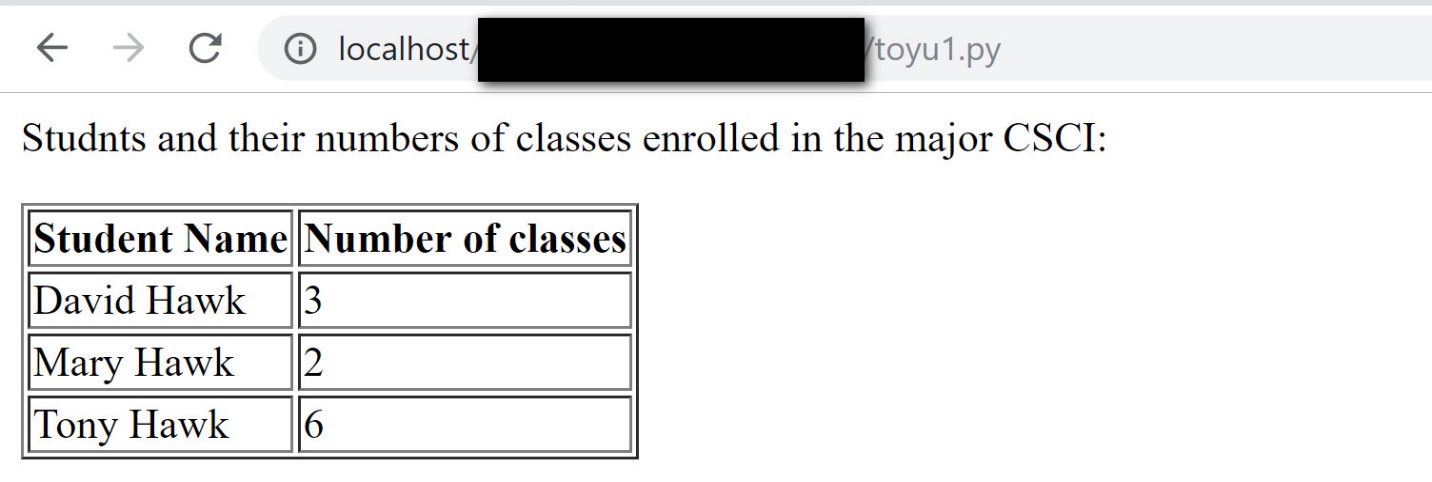
**4. Basic steps in Web DB development**

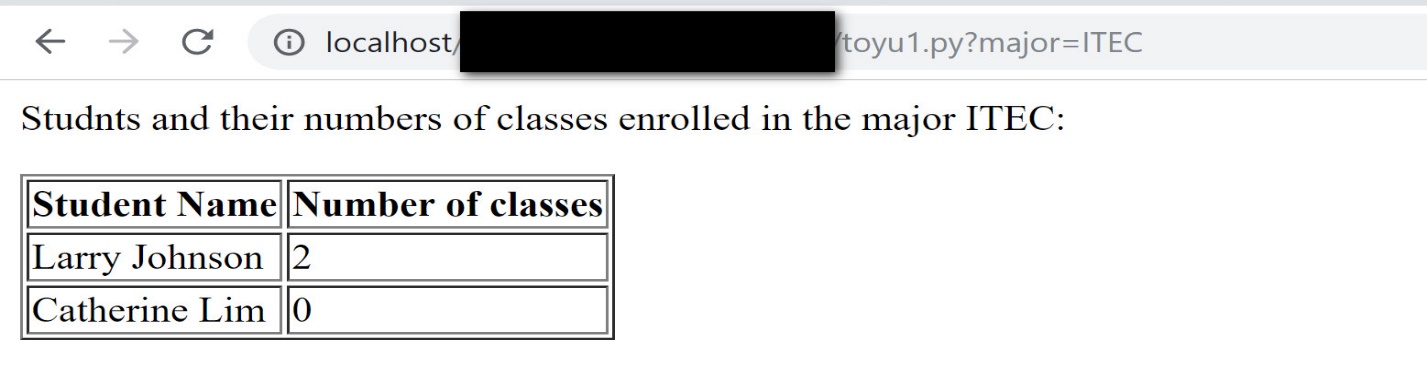
1. Identify the input parameters.
2. Develop the desirable Web page for typical input parameters.
3. Analyze the code of (2) and identify which parts are constants and which parts (*dynamic data*) should be generated from database.
4. Construct and test the SQL statements to fetch the dynamic data.
5. Write the Python code for (3) accordingly.

***Example:***

Write a Python CGI program, toyu1.py, to accept a HTTP Get parameter major and display the students majoring in major and their number of enrolled courses.

For example, for http://.../toyu1.py?major=CSCI:





There is no need for error checking of the user input parameter major. A skeleton for toyu1.py is provided for you.

**Solution:**

[1] Identify the input parameters.

major: typical values: 'CSCI' and 'ITEC'.

[2] Develop the desirable Web page for typical input parameters.

For CSCI:

<html>  
<head></head>  
<body>  
  
<p>Students and their numbers of classes enrolled in the major CSCI:</p>  
<table border='1'>  
<tr><th>Student Name</th><th>Number of classes</th></tr>  
  
<tr><td>David Hawk</td><td>3</td></tr>  
<tr><td>Mary Hawk</td><td>2</td></tr>  
<tr><td>Tony Hawk</td><td>6</td></tr>  
</body>  
</html>

For ITEC:  
  
<html>  
<head></head>  
<body>  
  
<p>Students and their numbers of classes enrolled in the major ITEC:</p>  
<table border='1'>  
<tr><th>Student Name</th><th>Number of classes</th></tr>  
  
<tr><td>Larry Johnson</td><td>2</td></tr>  
<tr><td>Catherine Lim</td><td>0</td></tr>  
</body>  
</html>

[3] Identify dynamic data: Analyze the code of (2) and identify which parts are constants and which parts (*dynamic data*) should be generated from database.

For CSCI:

<html>  
<head></head>  
<body>  
  
<p>Students and their numbers of classes enrolled in the major *CSCI*:</p>  
<table border='1'>  
<tr><th>Student Name</th><th>Number of classes</th></tr>  
  
<tr><td>*David Hawk*</td><td>*3*</td></tr>  
<tr><td>*Mary Hawk*</td><td>*2*</td></tr>  
<tr><td>*Tony Hawk*</td><td>*6*</td></tr>  
</body>  
</html>

For ITEC:  
  
<html>  
<head></head>  
<body>  
  
<p>Students and their numbers of classes enrolled in the major *ITEC*:</p>  
<table border='1'>  
<tr><th>Student Name</th><th>Number of classes</th></tr>  
  
<tr><td>*Larry Johnson*</td><td>*2*</td></tr>  
<tr><td>*Catherine Lim*</td><td>*0*</td></tr>  
</body>  
</html>

Dynamic Data:

* For each student, student.lname, student.fname, count of classId in the enroll table of the student.

[4] Construct and test the SQL to fetch the dynamic data.

for CSCI:

SELECT CONCAT(s.fName, ' ', s.lName) AS student,  
   COUNT(e.classId) as num\_classes  
FROM student AS s LEFT JOIN enroll AS e ON (s.stuId = e.stuId)  
WHERE s.major = *'CSCI'*  
GROUP BY s.stuId, student  
ORDER BY s.lName, s.fName;

for ITEC:

SELECT CONCAT(s.fName, ' ', s.lName) AS student,  
   COUNT(e.classId) as num\_classes  
FROM student AS s LEFT JOIN enroll AS e ON (s.stuId = e.stuId)  
WHERE s.major = *'ITEC'*  
GROUP BY s.stuId, student  
ORDER BY s.lName, s.fName;

The typical input will be replaced by a parameter in the Python/SQL code.

[5] Write the Python code for (3) accordingly.

[toyu1.py](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/web/toyu1.py.txt)

#!"c:\python310\python.exe"  
from dbconfig import \*  
import pymysql  
import cgi  
import cgitb  
cgitb.enable()  
  
print("Content-Type: text/html;charset=utf-8")  
print()  
  
print ('''<html>  
<head></head>  
<body>  
''')  
  
#   
db = get\_mysql\_param()  
cnx = pymysql.connect(user=db['user'],  
      password=db['password'],  
      host=db['host'],  
      # port needed only if it is not the default number, 3306.  
      # port = int(db['port']),  
      database=db['database'])  
cursor = cnx.cursor()  
  
form = cgi.FieldStorage()  
  
major = form.getfirst('major')  
if major is None:  
    print ('Please enter a valid major code in the URL.')  
    print ('</body></html>')  
    quit()  
  
query = '''  
SELECT CONCAT(s.fName, ' ', s.lName) AS student,  
COUNT(e.classId) as num\_classes  
FROM toyu.student AS s LEFT JOIN toyu.enroll AS e ON (s.stuId = e.stuId)  
WHERE s.major = %s  
GROUP BY s.stuId, student  
ORDER BY s.lName, s.fName;  
'''  
  
cursor.execute(query,(major,))  
row = cursor.fetchone()  
  
if row is None:  
    print ('Sorry, no major with code {}. Please enter a valid major code in the URL.'.format(major))  
    print ('</body></html>')  
    quit()  
  
print('''<p>Studnts and their numbers of classes enrolled in the major {}:</p>  
<table border='1'>  
<tr><th>Student Name</th><th>Number of classes</th></tr>  
'''.format(major))  
  
while row is not None:  
    (student, count) = row  
    print("<tr><td>{}</td><td>{}</td></tr>".format(student, count))  
    row = cursor.fetchone()  
cursor.close()  
cnx.close()  
  
print ('''</body>  
</html>''')

**Some hints for debugging**

1. Run the Python program as a standalone program to ensure no syntax error.
2. For runtime error, check to see whether the HTTP header has sent to the Web server by the Python program.
3. Use log files provided by Apache.