**DASC 5333**

11/5/2024

Your Web-Python assignment needs to be uploaded to the DCM server for grading.

[1] A directory has been created for each student in DCM: fa24<<course>>s<<section in one digit>><<full last name>><<first character of first name>>. For examples:

|  |  |
| --- | --- |
| **Student** | **DCM directory for the student** |
| Bun Yue in CSCI 4333 Section 1 | fa24c4333s1yueb |
| Jane Jacob in CSCI 4333 Section 2 | fa24c4333s2jacobj |
| Mounika Patel in DASC 5333 Section 1 | fa24d5333s1patelm |

[2] Use FTP to upload the assignment (h6.py) to your directory.

* FTP server: dcm.uhcl.edu
* Login username and password: Your PCLAB account credential.

Filezilla:

A screenshot of a computer

Description automatically generated

URL: dcm.uhcl.edu/yue/temp/d5333/help.html

IN DCM server: /yue/pages/temp/d5333/help.html

[3] Your file(s) (e.g., h6.py) should be uploaded to the “pages” folder of your DCM directory, e.g., fa24c4333s1yueb/pages. It can then be accessed using the URL: http://dcm.uhcl.edu/fa24c4333s1yueb /h6.py.

[4] A MySQL guest account ‘dbguest’ (with the password ‘uhcl\_\_dbguest’) has been created with read privileges to the swim and toyu databases. Thus, your uploaded Python program should use the following credentials:

|  |
| --- |
| [mysql]  Host: localhost user: dbguest password: uhcl\_\_dbguest database: swim |

[5] If you have any questions about your DCM account, you may contact the student systems administrators: Charan Gedipudi or Harini Navari, Email: [gedipudic1346@uhcl.edu](mailto:gedipudic1346@uhcl.edu)

Email: [navarih8507@uhcl.edu](mailto:navarih8507@uhcl.edu). Please copy your email to me and the faculty system administrator (Abeysekera, Krishani at <Abeysekera@uhcl.edu>)

[6] For questions about the homework, you may ask my TA (Pavan Kodavali at [kodavali@uhcl.edu](mailto:kodavali@uhcl.edu)). Do not send questions about the assignments or the course materials to the systems administrators.

[7] For additional information about the DCM server, see: <http://sce.uhcl.edu/support/>.

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

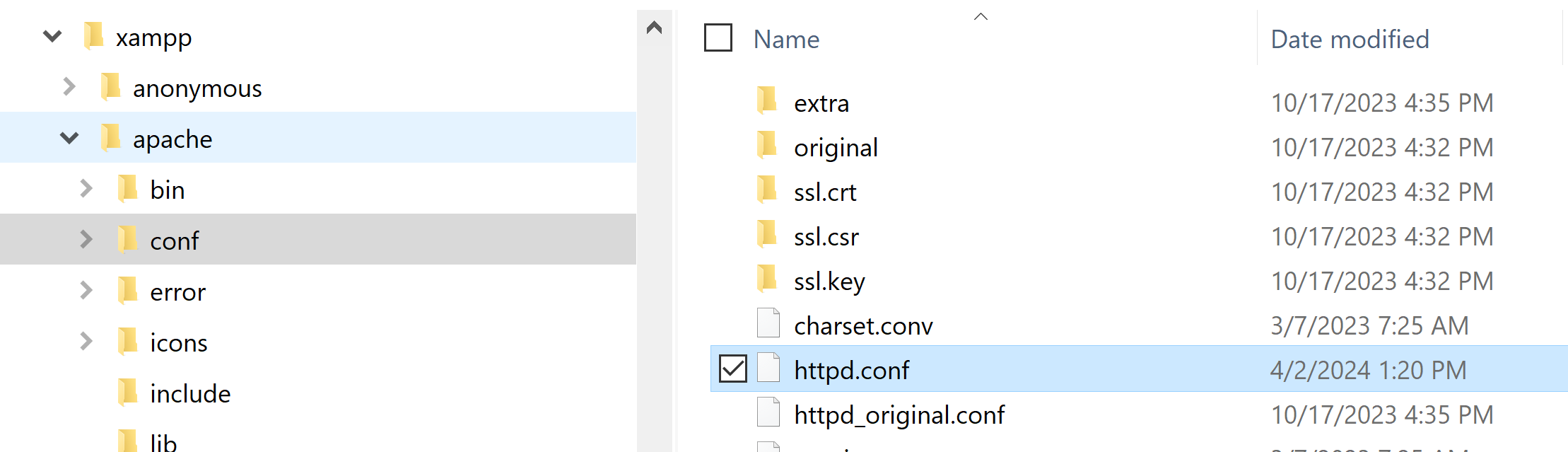
by K. Yue

**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.

Localhost:

Localhost/python/joindb/f2024/d5333/hello.html

Local file system C:\xampp\htdocs\python\joindb\f2024\d5333\hello.html

A diagram of a server

Description automatically generated

**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. (meta-data to the client)
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>  
''')

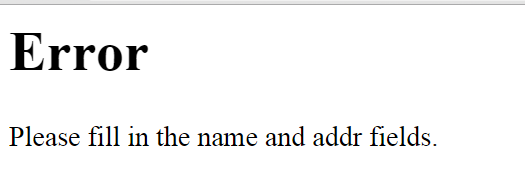
For: http://localhost/.../cgi1.py

or

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

or

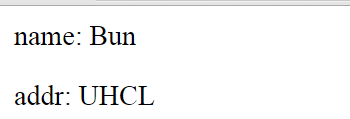
http://localhost/.../cgi1.py?Addr=UHCL



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.

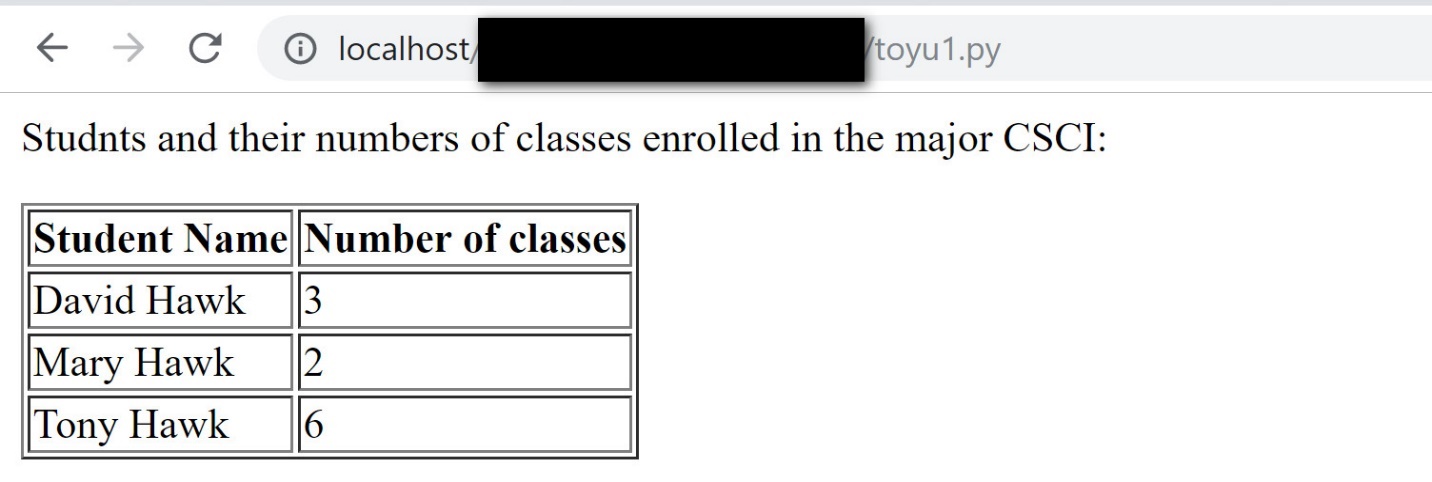
**4. Basic steps in Web DB development**

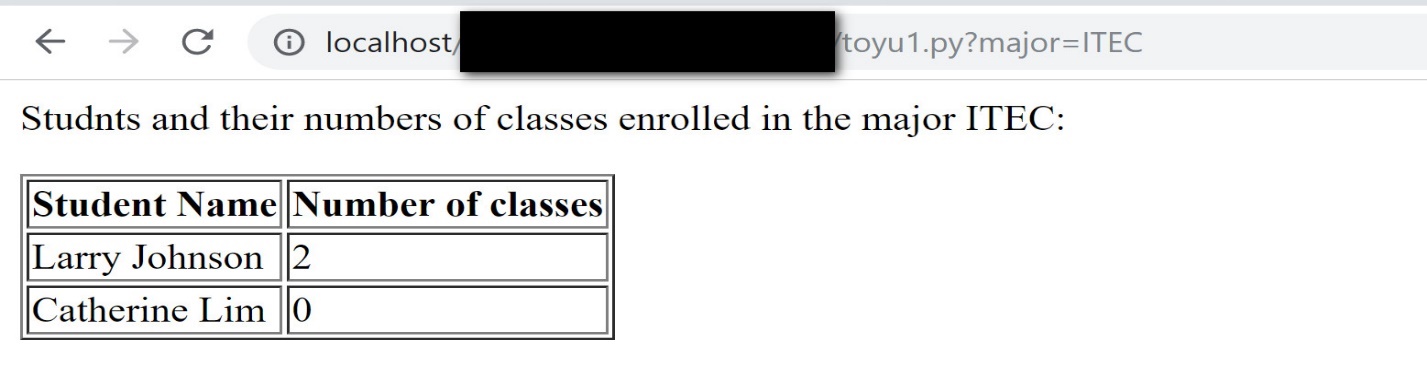
1. Identify the input parameters.
2. Develop the desirable Web page for typical input parameters.
3. Analyze the code of (2) and identify which parts are constants and which parts (*dynamic data*) should be generated from database.
4. Construct and test the SQL statements to fetch the dynamic data.
5.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

**Introduction to Database Design**

K. Yue

**1. Introduction**

* Bad database/table designs result in *unnecessary redundancy*: redundancy with limited or no benefit.
* However, redundancy can serve many purposes (security, integrity, error detection and correction, fault tolerance, performance, reliability), and are employed frequently in computer science.
* Problems:
  1. Inefficient storage
  2. Anomaly: data inconsistency, loss of data integrity, difficulties in maintenance.

***Example:***

Consider the vastly simplified and poorly-designed relation/table:

Employee\_Bad(EmpId, Name, DeptId, MailCode).

Assumptions made:

1. Every employee has an unique EmpId.
2. Every employee is represented as a tuple in the Employee relation.
3. Every employee works for only one department.
4. Every department has an unique DeptId.
5. Every department has exactly one mail code, stored in the field MailCode.

Thus, EmpId is a candidate key (CK). An instance of Employee\_Bad:

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **MailCode** |
| 101 | Lady Gaga | *D123* | *M10* |
| 122 | Brad Pitts | *D123* | *M10* |
| 140 | Lebron James | *D123* | *M10* |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| 311 | John Smiths | D300 | M33 |

**Problem:**

* Unnecessary redundancy: e.g., the fact that the mail code M10 for Department D123 is stored in three rows.

**1.1 Update Anomaly:**

(a) The mail code of department D123 is updated to M44.

* Inefficiency in update: needs to update all three rows in this example.
* Potential (logical data) inconsistency.

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **MailCode** |
| 101 | Lady Gaga | D123 | *M10 -> M44* |
| 122 | Brad Pitts | D123 | *M10 -> M44* |
| 140 | Lebron James | D123 | *M10 -> M44* |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| 311 | John Smiths | D300 | M33 |

SQL solution: it works but is not efficient.

UPDATE Employee  
SET MailCode = 'M44'  
WHERE DeptId = 'D123';

(b) Jennifer Lopez is reassigned to work for Department D300:

* Need to know the mail code of D300.
* Potential inconsistency.

The table may become:

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **MailCode** |
| 101 | Lady Gaga | D123 | M10 |
| 122 | Brad Pitts | D123 | M10 |
| 140 | Lebron James | D123 | M10 |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | **D300** | M21 *(may not be updated to M33)* |
| 311 | John Smiths | D300 | M33 |

The intuitive SQL command:

UPDATE Employee  
SET DeptId = 'D300'  
WHERE Name = 'Jennifer Lopez';

will produce inconsistent results, as shown in the table above.

One needs to update both DeptId and MailCode. However,

UPDATE Employee  
SET DeptId = 'D300',  
     MailCode = (SELECT DISTINCT MailCode FROM Employee WHERE DeptId = 'D300')  
WHERE Name = 'Jennifer Lopez';

will not work in MySQL as one cannot include a SELECT clause on the same table in the SET clause of an UPDATE statement.

A possible solution using a session variable, @mailCode:

SELECT DISTINCT MailCode INTO @mailCode  
FROM Employee WHERE DeptId = 'D300';

UPDATE Employee  
SET DeptId = 'D300',  
     MailCode = @mailCode  
WHERE Name = 'Jennifer Lopez';

**1.2 Insertion Anomaly:**

It is not possible creating a new Department D777, with the mail code M40 but no employee working for it yet. This is because, as the PK of Employee\_Bad, EmpId cannot be null.

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **MailCode** |
| 101 | Lady Gaga | D123 | M10 |
| 122 | Brad Pitts | D123 | M10 |
| 140 | Lebron James | D123 | M10 |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| 311 | John Smiths | D300 | M33 |
| ***????*** | **????** | **D777** | **M40 *(this row cannot be added)*** |

**1.3 Deletion Anomaly**

John Smiths no longer works here. Result: the information that M33 is the mail code of Department D300 is also lost.

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **MailCode** |
| 101 | Lady Gaga | D123 | M10 |
| 122 | Brad Pitts | D123 | M10 |
| 140 | Lebron James | D123 | M10 |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| *~~311~~* | *~~John Smiths~~* | *~~D300~~* | *~~M33~~* |

**1.4 Decomposition**

A standard way of resolving unnecessary redundancy in poorly designed tables is by proper **decomposition**: breaking down a relation into two or more component relations.

***Example:*** the decomposition of the relation Employee\_Bad into two relations:

1. Empolyee(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | ~~D222~~ D300 |
| ~~311~~ | ~~John Smiths~~ | ~~D300~~ |

2. Department(DeptId, MailCode)

|  |  |
| --- | --- |
| **DeptId** | **MailCode** |
| D123 | ~~M10~~ M44 |
| D222 | M21 |
| D300 | M33 |
| D777 | M40 |

To obtain the original relation Employee\_Bad(EmpId, Name, DeptId, MailCode) from

Employee(EmpId, Name, DeptId)  
Department(DeptId, MailCode)

Relational algebra: using natural join, |x|.

Employee\_Bad = Employee |x| Department

This decomposition is said to be a *lossless* decomposition.

SQL:

SELECT Employee.\*, Department.MailCode  
FROM Employee INNER JOIN Department ON (Employee.DeptId = Department.DeptId);  
  
or  
  
SELECT \*  
FROM Employee NATURAL JOIN Department;

1. There is no loss of information: the definition of *lossless* decomposition.
2. No previously mentioned unnecessary redundancy and anomaly.

The actions of the four use cases that produced anomaly in Employee\_Bad:

1. Empolyee(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | *~~D222~~ D300* |
| *~~311~~* | *~~John Smiths~~* | *~~D300~~* |

2. Department(DeptId, MailCode)

|  |  |
| --- | --- |
| **DeptId** | **MailCode** |
| D123 | *~~M10~~ M44* |
| D222 | M21 |
| D300 | M33 |
| *D777* | *M40* |

**2. Methods for good database designs**

Two main tools:

1. Integrity Rules:  data constraint rules for avoiding data inconsistency.
2. Normal Forms:  a set of rules for designing good relation schemas.

**3. Integrity Rules**

**3.1 Database-Specific Integrity Rules**

* Most of the integrity rules are *application* dependent.
* Need to analyze the semantics of the applications to find out the integrity rules.
* These are known as Database-Specific Integrity Rules, or Application-Specific Integrity Rules.
  + Specific to an application
  + Not universally applicable.

***Examples***: some database-specific integrity rules.

1. Student Id should be a seven-digit number.
2. Date of Birth should be greater than 1900.
3. The room number of Delta Building should start with a 'D'.
4. A student cannot take more than 24 credits in any semester.
5. A student must show proof of a meningitis shot before registration for the first semester.

**3.2 General Integrity Rules**

* They should be satisfied by *every* database.
* However, they are not necessarily enforced by the DBMS.
* Two general integrity rules in relational databases:
  1. Entity Integrity Rule: based on the concepts of primary keys (PK) and candidate keys (CK)
  2. Referential Integrity Rule: based on the concept of foreign keys (FK).

**3.2.1 Entity Integrity Rule**

* Entity Integrity: no*component* of a *candidate key* of a relation can have a null value.
* Meaning: In a relational database, a row that cannot be identified by ites CK will not be stored.

***Example:***

Employee(EmpId, Name, DeptId, Salary)

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **Salary** |
| 101 | Lady Gaga | D123 | 55000000 |
| 122 | Brad Pitts | D123 | 10100000 |
| 140 | Lebron James | D123 | 50000000 |
| 155 | Narendra Modi | @: null | @ |
| *@* | Jennifer Lopez | D222 | 20000000***(should not be able to add this row)*** |
| *@* | John Smiths | D300 | 70000 ***(should not be able to add this row)*** |

* If EmpId is a candidate key, this Employee instance does not satisfy the entity integrity rule.
* Conversely, if we accept the relation instance above as valid, EmpId cannot be a candidate key.
* Most DBMS enforce the entity integrity rule. DB developers just need to declare the primary keys (using PRIMARY) and the candidate key (using UNIQUE and not NULL) in their CREATE TABLE statements.

**3.2.2 Referential Integrity Rule**

* Referential integrity rule: relations should not contain any *unmatched* *non-null foreign key* values.
* Any non-null value of a foreign key K must appear in the parent (referenced) relation where K is a candidate key.

***Example:***

Employee(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | *@* |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | D222 |
| 311 | John Smiths | D300 |

Department(DeptId, MailCode)

|  |  |
| --- | --- |
| **DeptId** | **MailCode** |
| D123 | M10 |
| D222 | M21 |
| D300 | M33 |

* DeptId is a foreign key in the table EMP, referencing DeptId in the table Department. Employee(DeptId) references Department(DeptId).
* The referential integrity rule is satisfied.
* Note that DeptId may be null in Employee.

***Example:***

Employee(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | D222 |
| 311 | John Smiths | D300 |
| **350** | **Bun Yue** | **D119*(should not be added)*** |

Department(DeptId, MailCode)

|  |  |
| --- | --- |
| **DeptId** | **MailCode** |
| D123 | M10 |
| D222 | M21 |
| D300 | M33 |

* The referential integrity rule is not satisfied in the example above.

Note:

1. In practical DBMS, pay attention to where the referential integrity rule is enforced.
2. For example, in MySQL, only the INNODB data engine enforces the referential integrity rule.
3. If the DBMS does not enforce the referential integrity rule, it will be the task of the DB developers to do so.