Web Interfaces for Database Servers

CGI, MySQLdb, and Sockets

- glueing the connections with Python
- functions of the server: connect, count, and main
- development of the code for the client

Displaying all Records in HTML Table

- extending the web interface
- retrieving and packing records
- the client displays HTML table

Displaying Sorted Records in Order

- radio buttons in HTML form
- processing forms with CGI scripts
- updated code for server and two clients

MCS 275 Lecture 26
Programming Tools and File Management
Jan Verschelde, 14 March 2008

Web Interfaces for Database Servers

CGI, MySQLdb, and Sockets

glueing the connections with Python

functions of the server: connect, count, and main

development of the code for the client

Displaying all Records in HTML Table

extending the web interface

retrieving and packing records

the client displays HTML table

Displaying Sorted Records in Order

radio buttons in HTML form

processing forms with CGI scripts

updated code for server and two clients

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

CGI, MySQLdb, and Sockets

glued by Python scripts

Goal: build web interface to MySQL database.

Components:

1. server is Python script using `MySQLdb`
2. client is a CGI script: web interface

Example database: `OurPyFiles` with `scripts` table.

Steps in ***incremental*** development:

1. script to count number of records
2. server listens to one connection
   sends to client number of records
3. run client script first on command line
4. second version of client script
   writes plain text on web page

Counting Number of Records

recall use of MySQLdb

`scripts_count.py` prints number of records in table `scripts` of MySQL database `OurPyFiles`.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```

Counting Number of Records

recall use of MySQLdb

`scripts_count.py` prints number of records in table `scripts` of MySQL database `OurPyFiles`.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```

Counting Number of Records

recall use of MySQLdb

`scripts_count.py` prints number of records in table `scripts` of MySQL database `OurPyFiles`.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```

Counting Number of Records

recall use of MySQLdb

`scripts_count.py` prints number of records in table `scripts` of MySQL database `OurPyFiles`.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```

Counting Number of Records

recall use of MySQLdb

`scripts_count.py` prints number of records in table `scripts` of MySQL database `OurPyFiles`.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```

Counting Number of Records

recall use of MySQLdb

`scripts_count.py` prints number of records in table `scripts` of MySQL database `OurPyFiles`.

Requirements for a successful run:

1. MySQL must be started: `sudo mysqld_safe`
2. run as `sudo python scripts_count.py`

```
import MySQLdb
db = MySQLdb.connect(db='OurPyFiles')
cr = db.cursor()
q = 'select count(*) from scripts'
cr.execute(q)
r = cr.fetchone()
n = int(r[0])
print 'the number of scripts : %d' % n
```

Web Interfaces for Database Servers

Functions of the Server

structure of `scripts_server.py`

```
import MySQLdb
from socket import *
def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """

def count():
    """
    Returns the number of scripts.
    """

def main():
    """
    Accepts connection and sends #scripts.
    """
```

Functions of the Server

structure of `scripts_server.py`

```
import MySQLdb
from socket import *

def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """

def count():
    """
    Returns the number of scripts.
    """

def main():
    """
    Accepts connection and sends #scripts.
    """
```

Functions of the Server

structure of `scripts_server.py`

```
import MySQLdb
from socket import *

def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """

def count():
    """
    Returns the number of scripts.
    """

def main():
    """
    Accepts connection and sends #scripts.
    """
```

Functions of the Server

structure of `scripts_server.py`

```
import MySQLdb
from socket import *

def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """

def count():
    """
    Returns the number of scripts.
    """

def main():
    """
    Accepts connection and sends #scripts.
    """
```

Defining Network Connections

```
hostname = ''   # use any address
number = 11267  # number for the port
buffer = 80     # size of the buffer

def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(1)
    print 'server waits for connection'
    client, client_address = server.accept()
    print 'server accepted connection from ',\
       client_address
    return client, server
```

Defining Network Connections

```
hostname = ''   # use any address
number = 11267  # number for the port
buffer = 80     # size of the buffer

def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(1)
    print 'server waits for connection'
    client, client_address = server.accept()
    print 'server accepted connection from ',\
       client_address
    return client, server
```

Defining Network Connections

```
hostname = ''   # use any address
number = 11267  # number for the port
buffer = 80     # size of the buffer

def connect():
    """
    Returns client and server socket
    to communicate with one client.
    """
    server_address = (hostname, number)
    server = socket(AF_INET, SOCK_STREAM)
    server.bind(server_address)
    server.listen(1)
    print 'server waits for connection'
    client, client_address = server.accept()
    print 'server accepted connection from ',\
       client_address
    return client, server
```

Counting the Number of Records

```
def count():
    """
    Returns the number of scripts.
    """
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    q = 'select count(*) from scripts'
    cr.execute(q)
    r = cr.fetchone()
    n = int(r[0])
    return n
```

The Function main()

```
def main():
    """
    Accepts connection and sends #scripts.
    """
    client, server = connect()
    print 'server connects to database'
    nb = count()
    print 'server sends #scripts to client'
    data = str(nb)
    client.send(data)
    print 'count sent, closing off'
    server.close()
```

The Function main( )

```
def main():
    """
    Accepts connection and sends #scripts.
    """
    client, server = connect()
    print 'server connects to database'
    nb = count()
    print 'server sends #scripts to client'
    data = str(nb)
    client.send(data)
    print 'count sent, closing off'
    server.close()
```

The Function main( )

```
def main():
    """
    Accepts connection and sends #scripts.
    """
    client, server = connect()
    print 'server connects to database'
    nb = count()
    print 'server sends #scripts to client'
    data = str(nb)
    client.send(data)
    print 'count sent, closing off'
    server.close()
```

Web Interfaces for Database Servers

CGI, MySQLdb, and Sockets
- glueing the connections with Python
- functions of the server: connect, count, and main
- development of the code for the client

Displaying all Records in HTML Table
- extending the web interface
- retrieving and packing records
- the client displays HTML table

Displaying Sorted Records in Order
- radio buttons in HTML form
- processing forms with CGI scripts
- updated code for server and two clients

First Version of the Client

in the file `scripts_client.py`

```
from socket import *

hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer

server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = client.recv(buffer)
print 'client received \"' + data + '\"'

client.close()
```

The Client is Web Interface

CGI script `scripts_web.py`

```
#!/Library/Frameworks/.../bin/python
print "Content-Type: text/plain\n\n"

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer
server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = client.recv(buffer)
print 'Number of scripts : ' + data

client.close()
```

The Client is Web Interface

CGI script `scripts_web.py`

```
#!/Library/Frameworks/.../bin/python
print "Content-Type: text/plain\n\n"

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer
server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = client.recv(buffer)
print 'Number of scripts : ' + data

client.close()
```

The Client is Web Interface

CGI script `scripts_web.py`

```
#!/Library/Frameworks/.../bin/python
print "Content-Type: text/plain\n\n"

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer
server_address = (hostname, number)
client = socket(AF_INET, SOCK_STREAM)
client.connect(server_address)

print 'client is connected'
data = client.recv(buffer)
print 'Number of scripts : ' + data

client.close()
```

Web Interfaces for Database Servers

CGI, MySQLdb, and Sockets
- glueing the connections with Python
- functions of the server: connect, count, and main
- development of the code for the client

Displaying all Records in HTML Table
- extending the web interface
- retrieving and packing records
- the client displays HTML table

Displaying Sorted Records in Order
- radio buttons in HTML form
- processing forms with CGI scripts
- updated code for server and two clients

Displaying all Records on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records
   and makes HTML table to display

Synchronization is very important: ***for every send of the server, there must be a matching recv by the client!***

Displaying all Records on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records
   and makes HTML table to display

Synchronization is very important: ***for every send of the server, there must be a matching recv by the client!***

Displaying all Records on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records
   and makes HTML table to display

Synchronization is very important: ***for every send of the server, there must be a matching recv by the client!***

Displaying all Records on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records
   and makes HTML table to display

Synchronization is very important: ***for every send of the server, there must be a matching recv by the client!***

Displaying all Records on Web Page

extending the web interface

To see all records on a web page:

1. server sends number of records to client
2. client receives number of records
3. server sends all records to client
4. client receives all records
   and makes HTML table to display

Synchronization is very important: ***for every send of the server, there must be a matching recv by the client!***

Functions of the Server

in file `scripts_servdb.py`

```
def ConnectClient():
    """
    Returns client and server socket.
    """
def CountRecords(c):
    """
    Returns the #scripts, given cursor c.
    """
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
def PackTuple(t):
    """
    Packs data tuple as string.
    """
```

The Function main()

in the file `scripts_servdb.py`

```
def main():
    """
    Accepts connection and sends records.
    """
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    client, server = ConnectClient()
    client.send(str(nb))
    R = RetrieveRecords(cr)
    for i in range(0,len(R)):
        client.send(PackTuple(R[i]))
    server.close()
```

The Function main()

in the file `scripts_servdb.py`

```
def main():
    """
    Accepts connection and sends records.
    """
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    client, server = ConnectClient()
    client.send(str(nb))
    R = RetrieveRecords(cr)
    for i in range(0,len(R)):
        client.send(PackTuple(R[i]))
    server.close()
```

The Function main()

in the file `scripts_servdb.py`

```
def main():
    """
    Accepts connection and sends records.
    """
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    client, server = ConnectClient()
    client.send(str(nb))
    R = RetrieveRecords(cr)
    for i in range(0,len(R)):
        client.send(PackTuple(R[i]))
    server.close()
```

Web Interfaces for Database Servers

Retrieving Records and Packing Tuples

```
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items
    separated by colons.  Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*' '
    return r
```

Retrieving Records and Packing Tuples

```
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items
    separated by colons.  Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*' '
    return r
```

Retrieving Records and Packing Tuples

```
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items
    separated by colons.  Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*' '
    return r
```

Retrieving Records and Packing Tuples

```
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items
    separated by colons.  Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*' '
    return r
```

Retrieving Records and Packing Tuples

```
def RetrieveRecords(c):
    """
    Given cursor c, returns all records.
    """
    q = 'select * from scripts'
    c.execute(q)
    return c.fetchall()

def PackTuple(t):
    """
    Packs the tuple as string with items
    separated by colons.  Notice padding!
    """
    s = t[0] + '-' + str(int(t[1])) + ':'
    s = s + str(t[2]) + ':' + t[3] + ':'
    r = s + (buffer - len(s))*' '
    return r
```

Web Interfaces for Database Servers

Code for the Client

start of the file `scripts_showall.py`

```
#!/Library/Frameworks/.../bin/python

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer

def PrintHeader(title):
   """
   writes title and header of page
   """
   print """Content-type: text/html
<html>
<head>
<title>%s</title>
</head>
<body>""" % title
```

Code for the Client

start of the file `scripts_showall.py`

```
#!/Library/Frameworks/.../bin/python

from socket import *
hostname = 'localhost'  # on same host
number = 11267          # same port number
buffer = 80             # size of the buffer

def PrintHeader(title):
    """
    writes title and header of page
    """
    print """Content-type: text/html
<html>
<head>
<title>%s</title>
</head>
<body>""" % title
```

The Function main() in the Client

in the file `scripts_showall.py`

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    RetrieveTable(client,n)
    client.close()
```

The Function main() in the Client

in the file `scripts_showall.py`

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    RetrieveTable(client,n)
    client.close()
```

The Function main() in the Client

in the file `scripts_showall.py`

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    RetrieveTable(client,n)
    client.close()
```

Retrieving and displaying Records

in the file `scripts_showall.py`

```
def RetrieveTable(s,n):
    """
    Retrieves table of n records,
    using socket s to communicate.
    """
    print "<table>"
    for i in range(0,n):
        data = s.recv(buffer)
        d = data.split(':')
        print "<tr>"
        print "<td>%d</td>" % i
        print "<td>%s</td>" % d[0]
        print "<td>%s</td>" % d[1]
        print "<td>%s</td>" % d[2]
        print "</tr>"
    print "</table>"
```

Retrieving and displaying Records

in the file `scripts_showall.py`

```
def RetrieveTable(s,n):
    """
    Retrieves table of n records,
    using socket s to communicate.
    """
    print "<table>"
    for i in range(0,n):
        data = s.recv(buffer)
        d = data.split(':')
        print "<tr>"
        print "<td>%d</td>" % i
        print "<td>%s</td>" % d[0]
        print "<td>%s</td>" % d[1]
        print "<td>%s</td>" % d[2]
        print "</tr>"
    print "</table>"
```

Retrieving and displaying Records

in the file `scripts_showall.py`

```
def RetrieveTable(s,n):
    """
    Retrieves table of n records,
    using socket s to communicate.
    """
    print "<table>"
    for i in range(0,n):
        data = s.recv(buffer)
        d = data.split(':')
        print "<tr>"
        print "<td>%d</td>" % i
        print "<td>%s</td>" % d[0]
        print "<td>%s</td>" % d[1]
        print "<td>%s</td>" % d[2]
        print "</tr>"
    print "</table>"
```

Web Interfaces for Database Servers

Radio Buttons for Sort Order

This HTML form is stored in

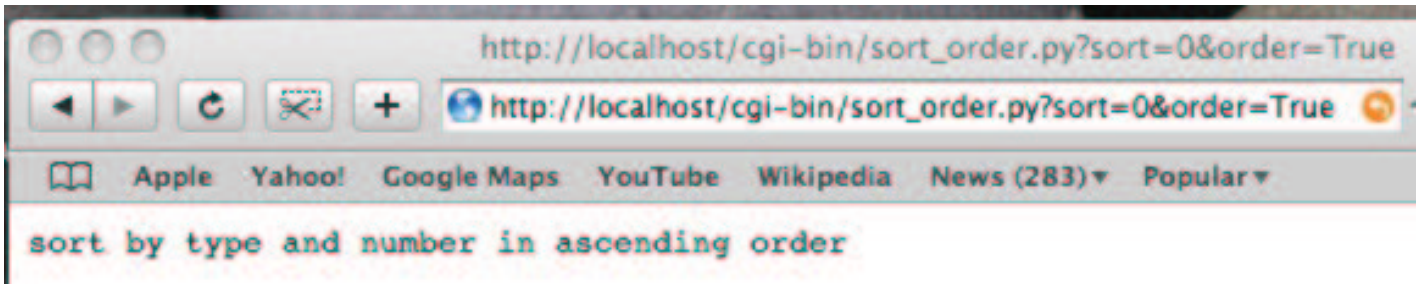
- users `public_html` directory on Unix
- users `Sites` directory on Mac OS X
- in `htdocs` of Apache directory on Windows

Radio Buttons for Sort Order

This HTML form is stored in

- users `public_html` directory on Unix
- users `Sites` directory on Mac OS X
- in `htdocs` of Apache directory on Windows

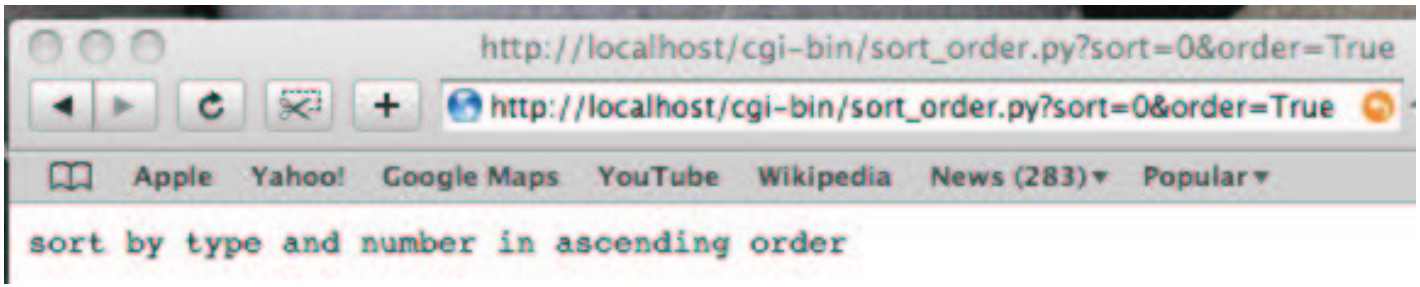
CGI Script to Confirm Choice



This CGI script is stored in

- `/var/www/cgi-bin` on Unix
- `/Library/WebServer/CGI-Executables` on Mac OS X
- `cgi-bin` on Apache directory on Windows

CGI Script to Confirm Choice



This CGI script is stored in

- `/var/www/cgi-bin` on Unix
- `/Library/WebServer/CGI-Executables` on Mac OS X
- `cgi-bin` on Apache directory on Windows

HTML Code

body in file `sort_order.html`

```
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by
<input type="radio" name="sort"
       value = 0 checked> type
<input type="radio" name="sort"
       value = 1> date
<input type="radio" name="sort"
       value = 2> name
<br>
order is
<input type="radio" name="order"
       value = True checked> ascending
<input type="radio" name="order"
       value = False> descending
</p>
<p> <input type="submit"> </p>
```

HTML Code

body in file `sort_order.html`

```
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by
<input type="radio" name="sort"
       value = 0 checked> type
<input type="radio" name="sort"
       value = 1> date
<input type="radio" name="sort"
       value = 2> name
<br>
order is
<input type="radio" name="order"
       value = True checked> ascending
<input type="radio" name="order"
       value = False> descending
</p>
<p> <input type="submit"> </p>
```

HTML Code

body in file `sort_order.html`

```
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by
<input type="radio" name="sort"
       value = 0 checked> type
<input type="radio" name="sort"
       value = 1> date
<input type="radio" name="sort"
       value = 2> name
<br>
order is
<input type="radio" name="order"
       value = True checked> ascending
<input type="radio" name="order"
       value = False> descending
</p>
<p> <input type="submit"> </p>
```

HTML Code

body in file `sort_order.html`

```
<h1> determine sort order </h1>
<form action="http://localhost/cgi-bin/sort_order.py">
<p>
sort by
<input type="radio" name="sort"
       value = 0 checked> type
<input type="radio" name="sort"
       value = 1> date
<input type="radio" name="sort"
       value = 2> name
<br>
order is
<input type="radio" name="order"
       value = True checked> ascending
<input type="radio" name="order"
       value = False> descending
</p>
<p> <input type="submit"> </p>
```

CGI Script

in file sort_order.py

```
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
   s = 'sort by type and number'
elif sortby == '1':
   s = 'sort by date'
else:
   s = 'sort by name'
if eval(orderis):
   s = s + ' in ascending order'
else:
   s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```

CGI Script

in file sort_order.py

```
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
   s = 'sort by type and number'
elif sortby == '1':
   s = 'sort by date'
else:
   s = 'sort by name'
if eval(orderis):
   s = s + ' in ascending order'
else:
   s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```

CGI Script

in file sort_order.py

```
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
   s = 'sort by type and number'
elif sortby == '1':
   s = 'sort by date'
else:
   s = 'sort by name'
if eval(orderis):
   s = s + ' in ascending order'
else:
   s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```

CGI Script

in file sort_order.py

```
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
   s = 'sort by type and number'
elif sortby == '1':
   s = 'sort by date'
else:
   s = 'sort by name'
if eval(orderis):
   s = s + ' in ascending order'
else:
   s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```

CGI Script

in file sort_order.py

```
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
   s = 'sort by type and number'
elif sortby == '1':
   s = 'sort by date'
else:
   s = 'sort by name'
if eval(orderis):
   s = s + ' in ascending order'
else:
   s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```

CGI Script

in file sort_order.py

```
#!/Library/Frameworks/.../bin/python
# L-26 MCS 275 Fri 14 Mar 2008 : sort_order.py
import cgi
form = cgi.FieldStorage()
sortby = form['sort'].value
orderis = form['order'].value
if sortby == '0':
   s = 'sort by type and number'
elif sortby == '1':
   s = 'sort by date'
else:
   s = 'sort by name'
if eval(orderis):
   s = s + ' in ascending order'
else:
   s = s + ' in descending order'
print "Content-Type: text/plain\n"
print s
```

Web Interfaces for Database Servers

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records,
   prints the form for the sort order, and
   activates the second client
2. second client processes the form,
   sends sort order to server, and
   retrieves and displays sorted records

Both clients after connection receive
the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records,
   prints the form for the sort order, and
   activates the second client
2. second client processes the form,
   sends sort order to server, and
   retrieves and displays sorted records

Both clients after connection receive
the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Processing Forms with CGI Scripts

how to do it

Good for testing:

1. `f.html` has form, action refers to `f.py`
2. `f.py` defines CGI script, invoked by `submit`

Integrated approach: Python scripts printing HTML.

Database server listens to two clients:

1. first client displays number of records, prints the form for the sort order, and activates the second client
2. second client processes the form, sends sort order to server, and retrieves and displays sorted records

Both clients after connection receive the number of records in the table.

Web Interfaces for Database Servers

main() in Server scripts_sortdb.py

```
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order \"' + sortorder + '\"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
```

main() in Server scripts_sortdb.py

```
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order \"' + sortorder + '\"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
```

main() in Server scripts_sortdb.py

```
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order \"' + sortorder + '\"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
```

main() in Server scripts_sortdb.py

```
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order \"' + sortorder + '\"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
```

main() in Server scripts_sortdb.py

```
def main():
    db = MySQLdb.connect(db='OurPyFiles')
    cr = db.cursor()
    nb = CountRecords(cr)
    sortclient, server = ConnectClient()
    sortclient.send(str(nb))
    print 'wait for submit client'
    submitclient, adr = server.accept()
    print 'submit client is connected'
    submitclient.send(str(nb))
    sortorder = submitclient.recv(buffer)
    print 'received sort order \"' + sortorder + '\"'
    R = RetrieveRecords(cr,sortorder)
    print 'sending records ...'
    for i in range(0,len(R)):
        submitclient.send(PackTuple(R[i]))
    print 'closing connection'
    server.close()
```

main() in First Client

in file scripts_sort.py

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('sorting all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    PromptSortOrder()
    client.close()
```

main() in First Client

in file scripts_sort.py

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('sorting all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    PromptSortOrder()
    client.close()
```

main() in First Client

in file scripts_sort.py

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('sorting all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<B>Number of scripts : %d</B>" % n
    PromptSortOrder()
    client.close()
```

main() in Second Client

in file scripts_sortall.py

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<b>Number of scripts : %d</b>" % n
    SendSortOrder(client)
    RetrieveTable(client,n)
    client.close()
```

main() in Second Client

in file scripts_sortall.py

```
def main():
    """
    Connects and prints data of server.
    """
    PrintHeader('showing all scripts')
    server_address = (hostname, number)
    client = socket(AF_INET, SOCK_STREAM)
    client.connect(server_address)
    data = client.recv(buffer)
    n = int(data)
    print "<b>Number of scripts : %d</b>" % n
    SendSortOrder(client)
    RetrieveTable(client,n)
    client.close()
```

First Client prompts Sort Order

```
def PromptSortOrder():
    """
    Display a form to ask user for
    field to sort on and the order.
    """
    print """
<form
 action="http://localhost/cgi-bin/scripts_sortall.py"
<p>
      ... rest of html code ...
"""
```

Second Client sends Sort Order

```
def SendSortOrder(cs):
    """
    Sends sort order to server
    using the client socket cs.
    """
    form = cgi.FieldStorage()
    sortby = form['sort'].value
    if eval(form['order'].value):
        sortby = sortby + '+'
    else:
        sortby = sortby + '-'
    cs.send(sortby)
```

Second Client sends Sort Order

```
def SendSortOrder(cs):
    """
    Sends sort order to server
    using the client socket cs.
    """
    form = cgi.FieldStorage()
    sortby = form['sort'].value
    if eval(form['order'].value):
        sortby = sortby + '+'
    else:
        sortby = sortby + '-'
    cs.send(sortby)
```

Second Client sends Sort Order

```
def SendSortOrder(cs):
    """
    Sends sort order to server
    using the client socket cs.
    """
    form = cgi.FieldStorage()
    sortby = form['sort'].value
    if eval(form['order'].value):
        sortby = sortby + '+'
    else:
        sortby = sortby + '-'
    cs.send(sortby)
```

Query in Server Script

```
def RetrieveRecords(c,sortorder):
    """
    Given cursor c, returns all records,
    taking sortorder into account.
    """
    q = 'select * from scripts'
    if sortorder[0] == '0':
       q = q + ' order by t,n'
    elif sortorder[0] == '1':
       q = q + ' order by d'
    else:
       q = q + ' order by f'
    if sortorder[1] == '+':
       q = q + ' asc'
    else:
       q = q + ' desc'
    c.execute(q)
    return c.fetchall()
```

Query in Server Script

```
def RetrieveRecords(c,sortorder):
    """
    Given cursor c, returns all records,
    taking sortorder into account.
    """
    q = 'select * from scripts'
    if sortorder[0] == '0':
        q = q + ' order by t,n'
    elif sortorder[0] == '1':
        q = q + ' order by d'
    else:
        q = q + ' order by f'
    if sortorder[1] == '+':
        q = q + ' asc'
    else:
        q = q + ' desc'
    c.execute(q)
    return c.fetchall()
```

Query in Server Script

```
def RetrieveRecords(c,sortorder):
    """
    Given cursor c, returns all records,
    taking sortorder into account.
    """
    q = 'select * from scripts'
    if sortorder[0] == '0':
        q = q + ' order by t,n'
    elif sortorder[0] == '1':
        q = q + ' order by d'
    else:
        q = q + ' order by f'
    if sortorder[1] == '+':
        q = q + ' asc'
    else:
        q = q + ' desc'
    c.execute(q)
    return c.fetchall()
```

Summary + Exercises

We covered chapter 12 in *Making Use of Python*, introducing web interfaces to database servers. Python glues CGI, MySQLdb, and Sockets *incrementally*. Exercises:

1. Provide a web interface to enter data in the table `scripts`. Use an HTML page to enter all data where the submit will activate a client of the database server. The client sends the user data to the server, the server adds it and sends feedback to the client.
2. Use tables `typedate` and `filedata` of Lecture 23 to make a web interface to retrieve records based on keys. Start at an HTML page with an input element to enter a key. The action in the form launches a client of the database server. The server retrieves the record and sends the data to the client for display.