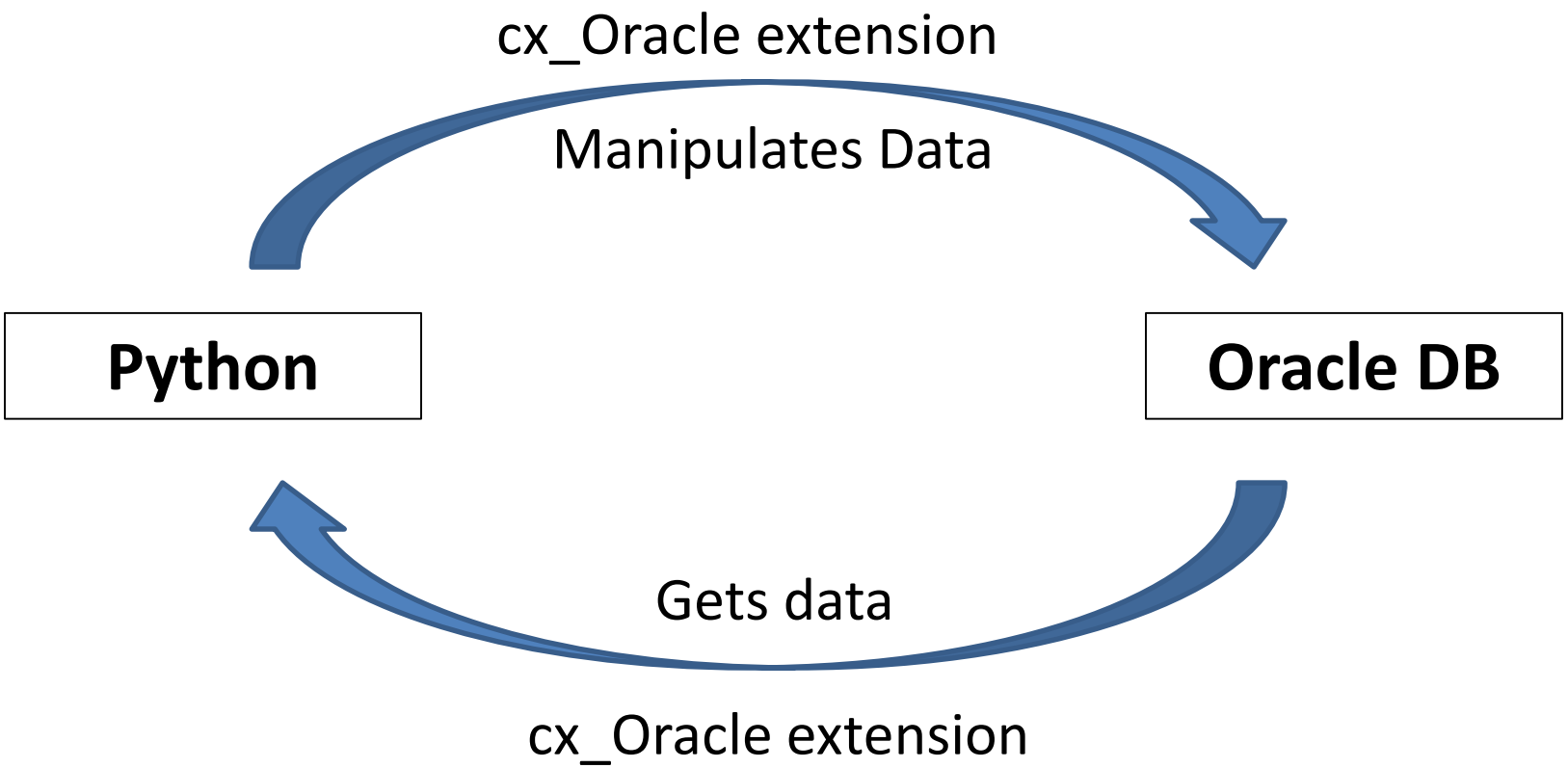
STSCI 4060

Lecture File 10

Xiaolong Yang
(xy44@cornell.edu)

Integrate Python with Oracle DB

Python and Oracle



Install cx_Oracle 7.x Extension

- **cx_Oracle** is a Python extension module for accessing Oracle databases and conforming to the Python database API specifications.
- This module is currently built against Oracle 11.2, 12.1, 12.2 and 18.3 and Python 2.7 as well as Python 3.5, 3.6 and 3.7.
- Go to https://oracle.github.io/python-cx_Oracle/ install the extension.
- For most cases, you just need to run the following command at your DOS shell:

python -m pip install cx_Oracle --upgrade

Note: in case of Spyder, if cx_Oracle is not installed, go to **Environments** and select **Not installed** and then **cx_Oracle** to install and upgrade. You may see an older version of cx_Oracle but it will work well.

The Oracle Database

- The most popular relational database system in the market/industry.
- It was introduced in STSCI 5060.
- It can integrate well with the SAS software for statistical data analysis.
- It can also integrate well with Python for general programming purposes, including data analyses.
- …

Install Oracle 11g XE

- If you do not have a local version of Oracle 11g XE, download it from http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html based on your computer's OS and install it.

- Also, download **Oracle SQL Developer** from the same website and install it.

Create an Oracle User Account

- While you are logged in as an Oracle **system user**, you have the privilege to create a user.
- Use your first name as the new database user name (if it was not been created before) and assign a password of your choice (write it down).
- Enter the following statement:
  CREATE USER *first_name* IDENTIFIED BY *your_password*;
  Then it displays "User created."
- Grant privileges to the new user:
  GRANT ALL PRIVILEGES TO *first_name;*
  Then it should display "Grant succeeded."
- Exit SQL by typing "exit" or "quit."
- Enter "sqlplus" at the prompt and log on with your new user name and password.
- You can check to see who the user is by typing the "show user" command.

Create a Table in Your New Database

One way to create a table is based on an existing table in other databases. The following statement creates a table called DEPARTMENTS from the DEPARTMENTS table in the HR database.

```
CREATE TABLE departments AS
        SELECT * FROM hr.departments;
```

You can show your table contents with the following query:

```
SELECT * FROM departments;
```

```
SQL> select * from departments;

DEPARTMENT_ID DEPARTMENT_NAME                MANAGER_ID LOCATION_ID
------------- ------------------------------ ---------- -----------
           10 Administration                        200        1700
           20 Marketing                             201        1800
           30 Purchasing                            114        1700
           40 Human Resources                       203        2400
           50 Shipping                              121        1500
           60 IT                                    103        1400
           70 Public Relations                      204        2700
           80 Sales                                 145        2500
           90 Executive                             100        1700
          100 Finance                               108        1700
          110 Accounting                            205        1700
```

Connect Python to Oracle

- First, you import the cx_Oracle module to provide the API for accessing your Oracle database.
- Then, you create an object "con" for a specific connection, in which the username and password are passed via the connect() method.
- The "con" object has a "version" attribute, which tells you the version of Oracle you are using.
- The close() method is called to close the connection.

At Python Shell, enter the following:

```
>>> import cx_Oracle
>>> con=cx_Oracle.connect('python/welcome')
 >>> print con.version
11.2.0.2.0
 >>> con.close()
```

Create a Simple Query in Oracle

query.py - C:\Python27\Oracle\query.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('select * from departments order by department_id')
for result in cur:
    print result

cur.close()
con.close()
```

Ln: 11 Col: 11

- The cursor() method opens a cursor for statements to use; a cursor is a pointer used to fetch rows from a result set.
- The execute() method parses and executes the SQL statement.
- The for loop fetches each row from the cursor and prints it.
- At the end, you need to close both the cursor and the connection.

The Result of a Simple Query in Oracle

Python Shell

File Edit Shell Debug Options Windows Help

```
>>>
(10, 'Administration', 200, 1700)
(20, 'Marketing', 201, 1800)
(30, 'Purchasing', 114, 1700)
(40, 'Human Resources', 203, 2400)
(50, 'Shipping', 121, 1500)
(60, 'IT', 103, 1400)
(70, 'Public Relations', 204, 2700)
(80, 'Sales', 145, 2500)
(90, 'Executive', 100, 1700)
(100, 'Finance', 108, 1700)
(110, 'Accounting', 205, 1700)
(120, 'Treasury', None, 1700)
(130, 'Corporate Tax', None, 1700)
(140, 'Control And Credit', None, 1700)
(150, 'Shareholder Services', None, 1700)
(160, 'Benefits', None, 1700)
(170, 'Manufacturing', None, 1700)
(180, 'Construction', None, 1700)
(190, 'Contracting', None, 1700)
(200, 'Operations', None, 1700)
(210, 'IT Support', None, 1700)
(220, 'NOC', None, 1700)
(230, 'IT Helpdesk', None, 1700)
(240, 'Government Sales', None, 1700)
(250, 'Retail Sales', None, 1700)
(260, 'Recruiting', None, 1700)
(270, 'Payroll', None, 1700)
>>>
```

Ln: 40 Col: 4

All the rows of the database table are fetched and are printed/displayed as Python **tuples**, row by row , one at a time.

Fetch Data From Database One Row a Time

query_one.py - C:\Python27\Oracle\query_one.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('select * from departments order by department_id')
row = cur.fetchone()
print row
row = cur.fetchone()
print row

cur.close()
con.close()
```

- The fetchone() method is used to return just a single row as a tuple.
- When the method is called multiple times, consecutive rows are returned.

Python Shell

File Edit Shell Debug Options Windows Help

```
>>>
(10, 'Administration', 200, 1700)
(20, 'Marketing', 201, 1800)
>>>
```

Ln: 7 Col: 4

Fetch a Specified Number of Rows From a Database

query_many.py - C:\Python27\Oracle\query_many.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('select * from departments order by department_id')
res = cur.fetchmany(numRows=4)
print res

cur.close()
con.close()
```

Ln: 3 Col: 31

- The fetchmany() method returns a list of tuples, each of which is a row.
- The numRows (here=4) parameter specifies the number of rows you want to return (counted from the beginning).

Python Shell

File Edit Shell Debug Options Windows Help

```
===========
>>>
[(10, 'Administration', 200, 1700), (20, 'Marketing', 201, 1800),
(30, 'Purchasing', 114, 1700), (40, 'Human Resources', 203, 2400)]
>>>
```

Ln: 6 Col: 4

Fetch all the Rows From a Database With fetchall()

query_all.py - C:\Python27\Oracle\query_all.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('select * from departments order by department_id')
res = cur.fetchall()
print res

cur.close()
con.close()
```

- The fetchall() method returns all the rows in a list of tuples.

Python Shell

File Edit Shell Debug Options Windows Help

```
>>>
[(10, 'Administration', 200, 1700), (20, 'Marketing', 201, 1800), (30, 'Purchasi
ng', 114, 1700), (40, 'Human Resources', 203, 2400), (50, 'Shipping', 121, 1500)
, (60, 'IT', 103, 1400), (70, 'Public Relations', 204, 2700), (80, 'Sales', 145,
2500), (90, 'Executive', 100, 1700), (100, 'Finance', 108, 1700), (110, 'Account
ing', 205, 1700), (120, 'Treasury', None, 1700), (130, 'Corporate Tax', None, 17
00), (140, 'Control And Credit', None, 1700), (150, 'Shareholder Services', None
, 1700), (160, 'Benefits', None, 1700), (170, 'Manufacturing', None, 1700), (180
, 'Construction', None, 1700), (190, 'Contracting', None, 1700), (200, 'Operatio
ns', None, 1700), (210, 'IT Support', None, 1700), (220, 'NOC', None, 1700), (23
0, 'IT Helpdesk', None, 1700), (240, 'Government Sales', None, 1700), (250, 'Ret
ail Sales', None, 1700), (260, 'Recruiting', None, 1700), (270, 'Payroll', None,
1700)]
```

Ln: 6 Col: 4

Fetch all the Rows From a Database With fetchall(), a slight modification

query_all_sep.py - C:\Python27\Oracle\query_all_sep.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('pythonhol/welcome')

cur = con.cursor()
cur.execute('select * from departments order by department_id')
res = cur.fetchall()
for r in res:
    print r

cur.close()
con.close()
```

Ln: 13 Col: 0

Python Shell

File Edit Shell Debug Options Windows Help

```
(10, 'Administration', 200, 1700)
(20, 'Marketing', 201, 1800)
(30, 'Purchasing', 114, 1700)
(40, 'Human Resources', 203, 2400)
(50, 'Shipping', 121, 1500)
(60, 'IT', 103, 1400)
(70, 'Public Relations', 204, 2700)
(80, 'Sales', 145, 2500)
(90, 'Executive', 100, 1700)
(100, 'Finance', 108, 1700)
(110, 'Accounting', 205, 1700)
(120, 'Treasury', None, 1700)
(130, 'Corporate Tax', None, 1700)
(140, 'Control And Credit', None, 1700)
(150, 'Shareholder Services', None, 1700)
(160, 'Benefits', None, 1700)
(170, 'Manufacturing', None, 1700)
(180, 'Construction', None, 1700)
(190, 'Contracting', None, 1700)
(200, 'Operations', None, 1700)
(210, 'IT Support', None, 1700)
(220, 'NOC', None, 1700)
(230, 'IT Helpdesk', None, 1700)
(240, 'Government Sales', None, 1700)
(250, 'Retail Sales', None, 1700)
(260, 'Recruiting', None, 1700)
(270, 'Payroll', None, 1700)
```

Ln: 32 Col: 4

The fetchall() method returns all the rows in a list of tuples; each of which is now printed separately. This is the same result as displayed on Slide 11.

Use Oracle's SQL Developer

Oracle SQL Developer

File Edit View Navigate Run Versioning Tools Help

Connecti... | 3n4.sql | rundong_yu_script_lab3n4.sql | stsci5060_project 2012fall.sql | query_arraysize.sql | python

Connecti

Cou

PVF

pyt

pyt

STS

New Connection...

Import Connections...

Export Connections...

Create Local Connections

Disconnect Connections

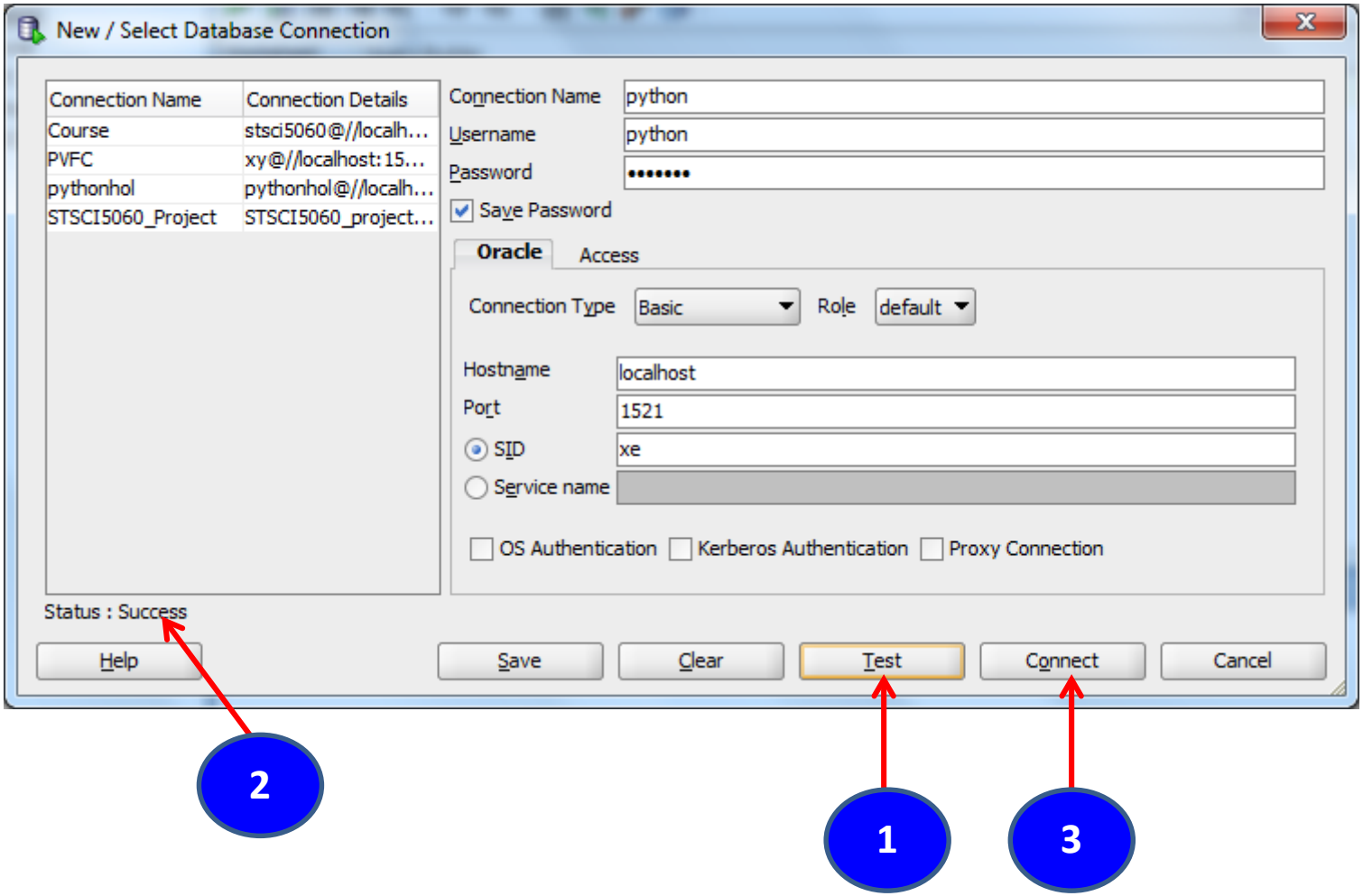
Query Builder

Messages - Log

```
2013-03-26 10:04:18 - Load Controllers
2013-03-26 10:04:20 - Init Recently opened Designs
```

Editing

Use Oracle's SQL Developer (cont'd)



Use Oracle's SQL Developer to Create a Big Table

Oracle SQL Developer : python

File Edit View Navigate Run Versioning Tools Help

Connections

- Connections
  - Course
  - PVFC
  - python
    - Tables (Filtered)
      - BIGTAB
        - SOMESTRIN
      - BIGTAB2
      - DEPARTMENTS
    - Views
    - Editioning Views
    - Indexes
    - Packages
    - Procedures
    - Functions
    - Queues
    - Queues Tables
    - Triggers

Start Page | stsci5060_project 2012fall.sql | query_arraysize.sql | BIGTAB | python

7.23899984 seconds

python

Worksheet | Query Builder

```
drop table bigtab;
create table bigtab (somestring varchar2(20));
begin
  for i in 1..20000
  loop
   insert into bigtab (somestring) values (dbms_random.string('A',20));
  end loop;
end;
/

commit;

exit
```

Return a string of mixed case alpha characters

Messages - Log

```
2013-03-26 10:04:18 - Load Controllers
2013-03-26 10:04:20 - Init Recently opened Designs
```

Line 11 Column 8 | Insert | Modified | Windows: CR/LF Editing

Improve Query Performance

Increase the number of rows returned in each batch from Oracle to the Python program can improve the query performance.

query_arraysize.py - C:\Python27\Oracle\query_arraysize.py

File Edit Format Run Options Windows Help

```
import time
import cx_Oracle

sizes=[1, 10, 100, 500, 800, 1000, 2000, 4000, 6000]

for aSize in sizes:
    con = cx_Oracle.connect('python/welcome')
    start = time.time()

    cur = con.cursor()
    cur.arraysize = aSize
    cur.execute('select * from bigtab')
    res = cur.fetchall()

    elapsed = (time.time() - start)
    print 'Array size =', aSize
    print 'The time elapsed is ', elapsed, "seconds.\n"

    cur.close()
    con.close()
```

- The 'time' module is used to measure elapsed time of the query.
- The arraysize is set to different values in **sizes** list. This causes batches of **arraysize** records at a time to be returned from the Oracle DB to a **cache** in Python, which reduces the number of "**roundtrips**" made to the DB, often reducing network load and reducing the number of <u>context switches</u> on the database server.
- The fetchall() method reads from the cache before requesting more data from Oracle.

Improve Query Performance (cont'd)

The output for comparing the performance

Python Shell

File Edit Shell Debug Options Windows Help

```
Array size = 1
The time elapsed is  0.842000007629 seconds.

Array size = 10
The time elapsed is  0.18700003624 seconds.

Array size = 100
The time elapsed is  0.0469999313354 seconds.

Array size = 500
The time elapsed is  0.0469999313354 seconds.

Array size = 800
The time elapsed is  0.0319998264313 seconds.

Array size = 1000
The time elapsed is  0.0320000648499 seconds.

Array size = 2000
The time elapsed is  0.0159997940063 seconds.

Array size = 4000
The time elapsed is  0.0310001373291 seconds.

Array size = 6000
The time elapsed is  0.0309998989105 seconds.

>>>
```

Ln: 136 Col: 4

Using Bind Variables

Bind variables enable you to re-execute statements with new values, without the overhead of **reparsing** the statement. As a result, it improves code reusability.

bind_query.py - D:/STSCI4060/STSCI4060_SP2013/Lectures/File7/bind_query.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.prepare('select * from departments where department_id = :id')

deptList = [10,20,30,210,110,180,260]
for deptid in deptList:
    cur.execute(None, {'id': deptid})
    res = cur.fetchone()
    print res

cur.close()
con.close()
```

Ln: 16

Python Shell

File Edit Shell Debug Options Windows Help

```
>>>
(10, 'Administration', 200, 1700)
(20, 'Marketing', 201, 1800)
(30, 'Purchasing', 114, 1700)
(210, 'IT Support', None, 1700)
(110, 'Accounting', 205, 1700)
(180, 'Construction', None, 1700)
(260, 'Recruiting', None, 1700)
>>>
```

Ln: 30 Col: 4

- Here the bind variable is ***:id***. The statement is only prepared once but executed seven times with different values for the WHERE clause.
- The special symbol '**None**' is used in place of the statement text argument to execute() because the prepare() method has already set the statement. The second argument to the execute() call is a Python dictionary.
- In the execute() calls, this dictionary has the values of the deptList.

Create a Table in Oracle Within Python

createEmptab.py - D:/STSCI4060/STSCI4060_SP2013/Lectures/File7/createEmptab.py

File Edit Format Run Options Windows Help

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')
cur=con.cursor()
cur.execute('create table emptab (id number, salary number, age number)')
cur.close()
con.close()
```

Ln: 8 Col: 0

Oracle SQL Developer : Table PYTHON.EMPTAB@python

File Edit View Navigate Run Versioning Tools Help

Connections

- Connections
  - Course
  - PVFC
  - python
    - Tables (Filtered)
      - BIGTAB
      - BIGTAB1
      - BIGTAB2
      - DEPARTMENTS
      - EMPTAB
      - MYTAB
    - Views

Start Page | stsci5060_project 2012fall.sql | EMPTAB | python~2

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes

Actions...

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID	NUMBER	Yes	(null)	1	(null)
SALARY	NUMBER	Yes	(null)	2	(null)
AGE	NUMBER	Yes	(null)	3	(null)

Messages - Log

```
2013-03-26 10:04:18 - Load Controllers
2013-03-26 10:04:20 - Init Recently opened Designs
```

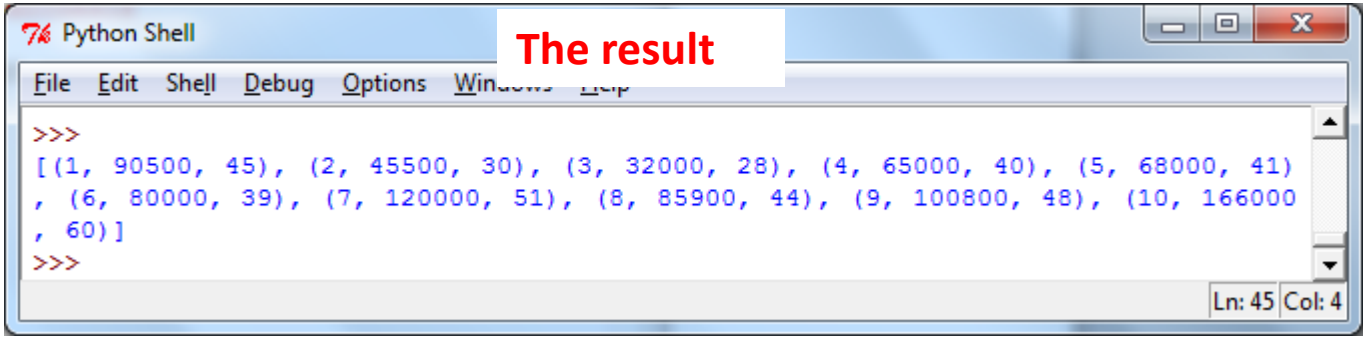
python | PYTHON | EMPTAB Editing

Populate the Oracle Table Within Python

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

rows = [ (1, 90500, 45),
         (2, 45500, 30),
         (3, 32000, 28),
         (4, 65000, 40),
         (5, 68000, 41),
         (6, 80000, 39),
         (7, 120000, 51),
         (8, 85900, 44),
         (9, 100800, 48),
         (10, 166000, 60)]

cur = con.cursor()
cur.bindarraysize = 10
cur.setinputsizes(int, 8, 3)
cur.executemany("insert into emptab(id, salary, age) values (:c1, :c2, :c3)", rows)

# Now query the results back
cur2 = con.cursor()
cur2.execute('select * from emptab')
res = cur2.fetchall()
print res

cur.close()
cur2.close()
con.close()
```

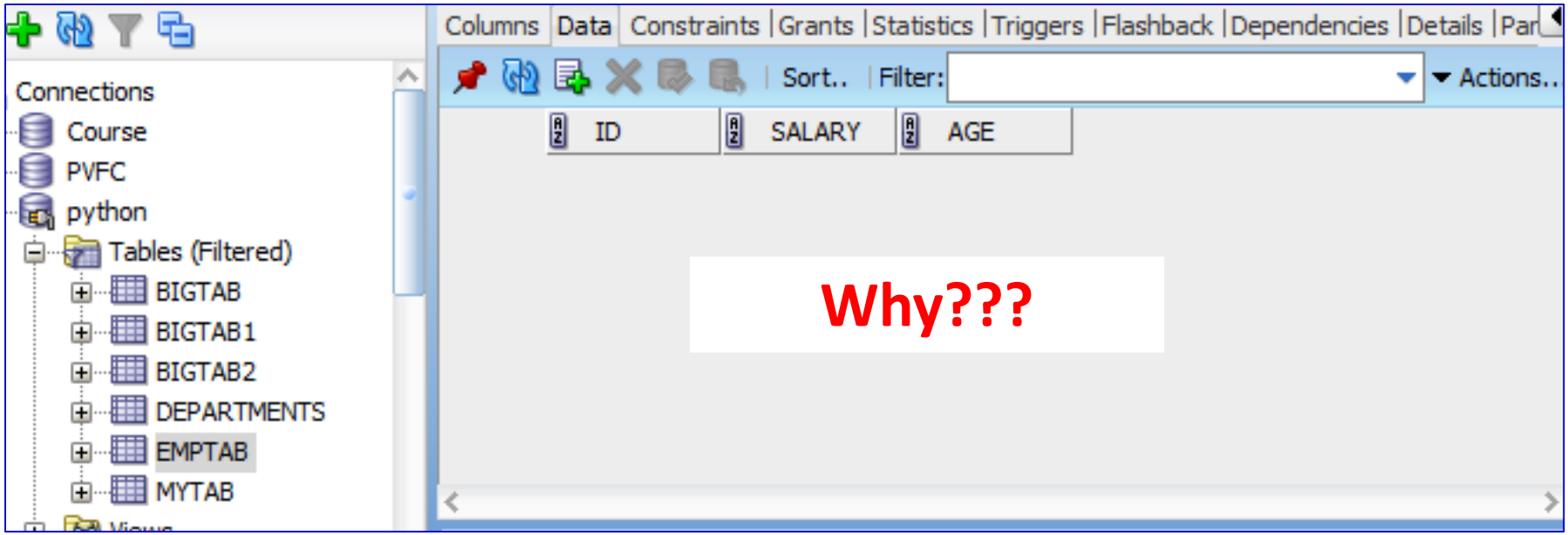
Populate the Oracle Table Within Python (cont'd)

You got a result in Python shell



Populate the Oracle Table Within Python (cont'd)

But you did not see the data in the Oracle table EMPTAB



Populate the Oracle Table Within Python (cont'd)

Because you did not commit the changes.
When you manipulate data in an Oracle database (insert, etc.), the changed or new data are only available within your database session until it is committed to the database by running the **commit()** method/command. When the changes are committed to the database, they are then available to other users and sessions. This is a database transaction. Note that the commit() method is used on the connection, not on the cursor.

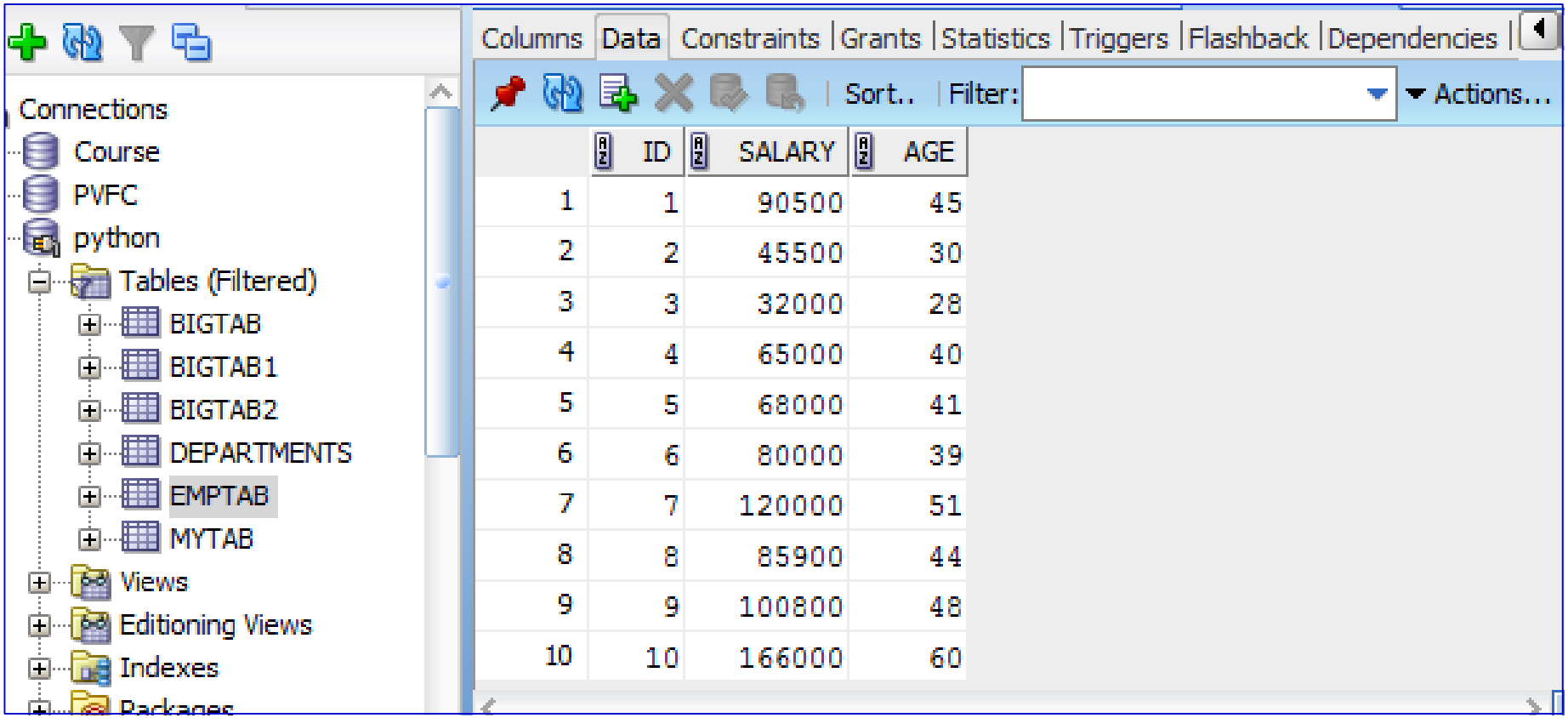
Populate the Oracle Table Within Python (cont'd)

Committing the changes – creating an transaction

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

rows = [ (1, 90500, 45),
         (2, 45500, 30),
         (3, 32000, 28),
         (4, 65000, 40),
         (5, 68000, 41),
         (6, 80000, 39),
         (7, 120000, 51),
         (8, 85900, 44),
         (9, 100800, 48),
         (10, 166000, 60)]

cur = con.cursor()
cur.bindarraysize = 10
cur.setinputsizes(int, 8, 3)
cur.executemany("insert into emptab(id, salary, age) values (:c1, :c2, :c3)", rows)

#Commit the changes to finish the DB transaction
con.commit()

# Now query the results back from the Oracle DB
cur2 = con.cursor()
cur2.execute('select * from emptab')
res = cur2.fetchall()
print res

cur.close()
cur2.close()
con.close()
```

Populate the Oracle Table Within Python (cont'd)

Now you can see the data in the Oracle table EMPTAB

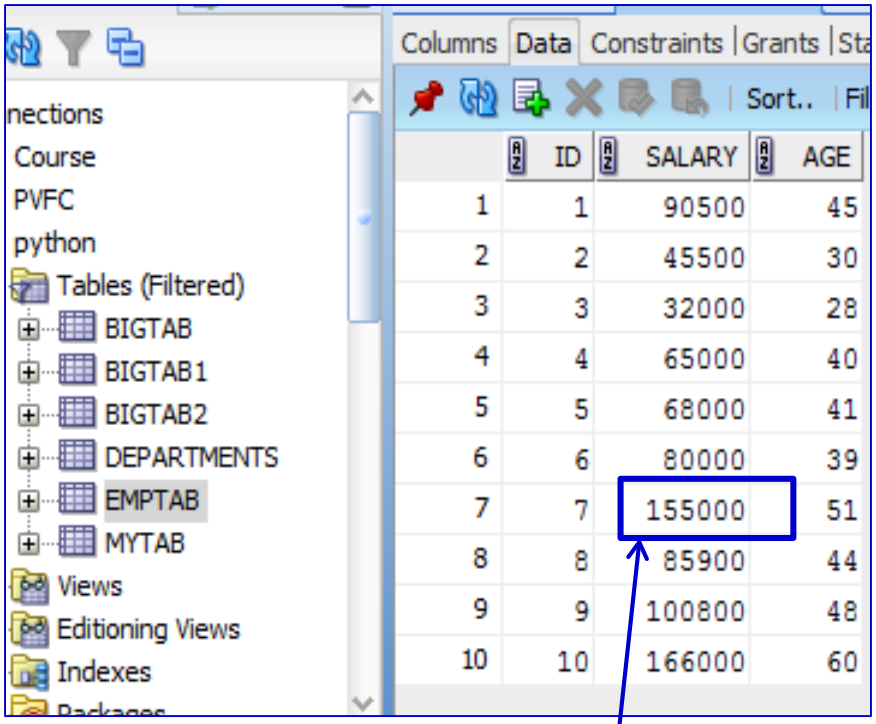


	ID	SALARY	AGE
1	1	90500	45
2	2	45500	30
3	3	32000	28
4	4	65000	40
5	5	68000	41
6	6	80000	39
7	7	120000	51
8	8	85900	44
9	9	100800	48
10	10	166000	60

Update an Oracle Table Through Python

Task: Raise the salary of the employee with id=7 from 120000 to 155000 due to a promotion in position.

```
import cx_Oracle

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('''update emptab
            set salary=155000
            where id=7''')

#Commit the changes to finish the DB transaction
con.commit()

# Now query the results back from the Oracle DB
cur2 = con.cursor()
cur2.execute('select * from emptab')
res = cur2.fetchall()
print res

cur.close()
cur2.close()
con.close()
```



The update made

Note: use triple quotation marks to keep the Oracle syntax format.

Obtain Data From the Oracle DB

Retrieve the salary data from the EMPTAB table in the Oracle DB and do some simple statistics of the data.

```
import cx_Oracle
import scipy

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('select salary from emptab')
sal=cur.fetchall()
num=range(len(sal))
print 'The salary list is:
for i in num:
    print '$' + str(sal[i][0]),
print '\nThe average salary is $' + str(scipy.mean(sal)) + '.'
print 'The maximum salary is $' + str(max(sal)[0]) + '.'
print 'The minimum salary is $' + str(min(sal)[0]) + '.'

cur.close()
con.close()
```

```
The salary list is:
$90500 $45500 $32000 $65000 $68000 $80000 $155000 $85900 $100800 $166000
The average salary is $88870.0.
The maximum salary is $166000.
The minimum salary is $32000.
```

Obtain Data From the Oracle DB, Another Example

Retrieve the salary and age data from the EMPTAB table in the Oracle DB and do a regression analysis by importing the Python program, mylr.py, which we wrote earlier in the class (with a small modification).

```
import cx_Oracle
import mylr

con = cx_Oracle.connect('python/welcome')

cur = con.cursor()
cur.execute('select salary, age  from emptab')
data=cur.fetchall()
num=range(len(data))
a=[]
s=[]
for i in num:
    s.append(data[i][0])
    a.append(data[i][1])

print 'The salary list (y variable): ', s
print 'The age list (x variable): ', a

print '\nThe linear regression result:'
mylr.lr(a,s)

cur.close()
con.close()
```

Obtain Data From the Oracle DB, Another Example