Database Systems (ISYS1001/ISYS5008)

Lecture 10

Connecting to a database via a programming language Interface(Python)

Please refer the previous year's slides regarding Java connectivity

Updated: 13th October,2021

Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

CRICOS Provide Code: 00301J

Copyright Warning

COMMONWEALTH OF AUSTRALIA
Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf of **Curtin University of Technology** pursuant to Part VB of the *Copyright Act 1968* (**the Act**)

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Learning outcomes

- Explain basic steps for connecting and using a database via a programming language interface.
- Write a python program to connect to MySQL server
- Write a python program to connect and then retrieve, insert, update data from the database
- NOTE : You can achieve learning outcomes using JAVA if you are from Java background.

Using the Database

- A database is generally used through a front-end user interface (UI), which is often windows-based.

- The UI needs to allow the following:
  - Execute common queries and display the results.
  - Execute common queries as reports and export them to file and/or print them.
  - Allow data to be inserted or updated.

Using the Database

- A database is generally used through a front-end user interface (UI), which is often windows-based.

- The UI needs to allow the following:
  - Execute common queries and display the results.
  - Execute common queries as reports and export them to file and/or print them.
  - Allow data to be inserted or updated.

UI Considerations

- The UI should attempt to validate and clean data before it is sent to the database.
  - Need to avoid common cyber attacks (such as SQL injection).
  - Ensure that data has the right format and meets other requirements.
- Remember, if your data gets corrupted then the database is close to useless.

UI Considerations

- Building the UI isn't part of this unit.
- Computing students will take the "Human Computer Interface" unit in 3rd year.
- For this unit, focus on the process of connecting the UI to the database.

Responsibilities: Front vs Back

- Team size & maintenance affects this.
- Front end needs to check all data being entered, as always.
  - There should never be a need to enter a semi-colon via the front end!
- The back end should manage all context-specific checks.
  - This includes checks for NULL and out of range values.
  - Also, be careful how strings are used in queries/commands.

Basic Safety

- Need to be aware of basic methods of securing your database against attacks.

- Some basic reading:
  - https://dev.mysql.com/doc/refman/5.7/en/security-against-attack.html
  - https://www.owasp.org/index.php/SQL_Injection
  - https://www.owasp.org/index.php/OWASP_Backend_Security_Project_MySQL_Hardening

Making the Connection

- The connection has the basic steps:
  1. Load appropriate libraries (if required).
  2. Connect to the database and log in.
  3. Set up the information/statement to send.
  4. Send the information/statement and receive reply if appropriate.
  5. Process the reply, if any.
  6. Disconnect from the database.

- You may repeat steps 3-5 several times before disconnecting.
- You can do this with fairly much any programming language:
  - We'll be looking at Python.
  - JAVA slides are available in the blackboard
  - Information on connecting with C is on Blackboard from previous years.

Python MySQL connection

- We will use the MySQL Connector for Python.
- Still relatively recent, and works for Python v2.7+ and v3.4+. This is the standard API and has good compatibility, but not necessarily optimal speed.
- On the VM, it is in the *yum packages mysql-connector-python* and *mysql-connector-python3*. Both are installed.
- Has features such as compression and encryption of communication.
- To install use the following in the command prompt :

```
pip install mysql-connector-python
```

Step 1: Libraries (and Registration)

- You need to *import mysql.connector*
- That's it for using Python.

Step 2: Connect to the Database

- We can use the following syntax:

```
conn =
  mysql.connector.connect(user='scott',
  password='tiger',
  host='127.0.0.1',
  database='company')
```

- Replace the appropriate details. Note that this also allows your to specify a database.
- Several other parameters are available too.
- Python doesn't have properties files as such, but has similar options if you want.
- *It's always god to use exception handling to give indication of errors and mange exceptions in connection set-up (and in other steps also).*

- NOTE : We can verify that the connection is working properly by typing following two commands in a **python console**, before proceeding with the python program:

```
> import mysql.connector
> mysql.connector.connect
(host='localhost',database='dswork',user='<user
name'>,password=<'password'>)
```

Step 3: Setting up Information

- We will be sending a statement or a query to the database.
- Expect the database to process it and send appropriate information back.
- The preparation required depends on what we want to do.

Example (python code):

```
select_stmt = "SELECT * FROM employees"
```

```
insert_stmt = (
        "INSERT INTO employees"
        "(emp_no,first_name, last_name, hire_date)"
        "VALUES (%s, %s, %s, %s)"
)
```

- *Examples are from MySQL API reference: https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html*

Step 4 : Send the query and get results if any

- In Python use *execute*, which is a property of the **cursor** class to execute the SQL statements.
- The cursor class allows a table to be stored and then traversed.
- An extension of execute is *executemany*, which executes the same operation repeatedly, using different items from a list each time.
- First, create a cursor object and then execute the statement using it.

Creating a cursor and execute it

- The *MySQLCursor* class instantiates objects that can execute operations such as SQL statements. Cursor objects interact with the MySQL server using a MySQLConnection object.
- Creating cursor():

```
import mysql.connector
conn = mysql.connector.connect(database=<'..‘>)
cursor = conn.cursor()
```

- Execute() executes the given database operation (query or command).

```
cursor.execute(operation,params=None,multi=False)
iterator = cursor.execute(operation,params=None,multi=True)
```

- The parameters found in the tuple or dictionary *params* are bound to the variables in the operation.
- Specify variables using %s or %(name)s parameter style (that is, using format or pyformat style).
- execute() returns an iterator if multi is True.

Example (python code):


```
select_stmt = "SELECT * FROM employees"
cursor.execute(select_stmt)
```

```
insert_stmt=("INSERT INTO employees (emp_no,first_name,last_name,hire_date)“
             "VALUES (%s, %s, %s, %s)" )
data1 = (2,'Jane','Doe', datetime.date(2012,3,23))
cursor.execute(insert_stmt, data1)
```

```
select_stmt = "SELECT * FROM employees WHERE emp_no = %(emp_no)s"
cursor.execute(select_stmt, { 'emp_no': 2 })
```

- *Examples are from MySQL API reference: https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html*

Example (python code):

```
data = [ ('Jane', date(2005, 2, 12)),
         ('Joe', date(2006, 5, 23)),
         ('John', date(2010, 10, 3)), ]

stmt = "INSERT INTO employees (first_name, hire_date) VALUES (%s, %s)"

cursor.executemany(stmt, data)
```

With the executemany() method, it is not possible to specify multiple statements to execute in the operation argument

Step 5: Process the reply, if any

- When the dataset is fetched, it can be processed iterating through records one by one.
- We can fetch all rows, several rows, next row etc. to a list using different commands (`fetchall(), fetchmany(), fetchone()`).
- Several other commands/ options are available.
- Then we can iterate through the list using python methods such as for loops.

Example: Fetching and processing

```
rows = cursor.fetchall()
```

```
cursor.execute("SELECT * FROM employees ORDER BY
                emp_no")
head_rows = cursor.fetchmany(size=2)
remaining_rows = cursor.fetchall()
```

- *Examples are from MySQL API reference: https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html*

Fetching and processing

```
# Using a while loop
cursor.execute("SELECT * FROM employees")
row = cursor.fetchone()
while row is not None:
    print(row)
    row = cursor.fetchone()
```

```
# Using the cursor as iterator
cursor.execute("SELECT * FROM employees")
for row in cursor:
        print(row)
        print("id:",row[0], " name: ", row[1])
```

- *Examples are from MySQL API reference: https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html*

Step 6: Disconnect from the database

- Close the cursor after fetching data
- This method closes the cursor, resets all results, and ensures that the cursor object has no reference to its original connection object.

```
cursor.close()
```

- Close the connecting after finishing all work with the database.

```
conn.close()
```

Example Program:


```
import datetime
import mysql.connector
Conn = mysql.connector.connect(user='scott',
database='company')
cursor = conn.cursor()

query = ("SELECT first_name, last_name, hire_date FROM
employees " "WHERE hire_date BETWEEN %s AND %s")

hire_start = datetime.date(2010, 1, 1)
hire_end = datetime.date(2010, 12, 31)

cursor.execute(query, (hire_start, hire_end))
for (first_name, last_name, hire_date) in cursor:
    print("{}, {} was hired on {:%d %b %Y}".format(
       last_name, first_name, hire_date))

cursor.close()
cnx.close()
```

Example from https://dev.mysql.com/doc/connector-python/en/connector-python-example-cursor-select.html

The Details

- What you need in order to satisfy the learning requirements for this unit is to know the process of connecting and sending statements and queries.
- This lecture skips some details of the code required but provides the foundation only.
- DDL statements (e.g. creating a table) also can be executed using the *programming language interface.
- The basic steps discussed here are applicable when connecting to other databases / other languages as well.

References and resources

- The information on the dev site is good:

https://dev.mysql.com/doc/connector-python/en/connector-python-introduction.html

- API from the MySQL reference (many examples in the lecture are taken from this link):

https://dev.mysql.com/doc/connector-python/en/connector-python-reference.html

- There are other decent tutorials:

http://www.mysqltutorial.org/getting-started-mysql-python-connector/

https://pynative.com/python-mysql-database-connection/

Summary

- Building a front-end is a standard part of database implementation.
- With some additional packages you can easily connect to a MySQL database via Python or other programming languages.
- After connection is established, you can then retrieve , insert, update and delete data using the SQL statements executed via the programming language constructs.
- Doing the actual front-end implementation with a GUI is not part of this unit.
- This material gives you the basics, and you'll learn more by yourself when you do this later in other projects.

Happy Database systems

Next week : Final lecture

Practical worksheet 10 (programming language connectivity)

Practical sheet will be available soon so you can try yourself to use it in the assignment.