Advanced SQL

IS-664 Database Programming
Fall 2022

hlocklear@pace.edu

Introduction To SQL Data Analysis

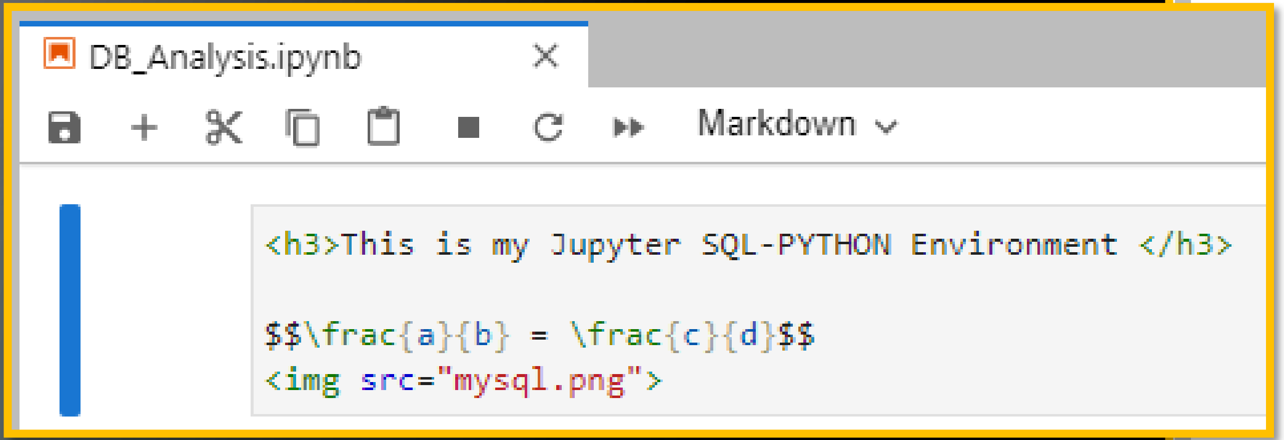
USE OF JUPYTER LAB WITH MYSQL

Professor HG Locklear

General

- The **Jupyter Lab** programming environment is a very powerful tool for **data analysis** and the **explanation of that analysis**.
- **Jupyter Lab** allows us to access our database(s) using **standard SQL commands** written in blocks or individual lines.
- Once we have our database data, we can then use the **Jupyter Lab** environment to manipulate it using many different languages (***we will only use Python***).
- In addition, **Jupyter Notebook** (part of Jupyter Lab) allows us to **document with text and images** the manipulation of our data.

General



We can include a wide variety of text, symbology, and graphics in our Jupyter Notebook.

SQL Code Block

We can use our **Jupyter Lab** environment to execute **SQL code blocks**.

```
%%sql
-- SQL Code Block
SELECT 'Hello' AS MSG
```

```
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

MSG
Hello

Cannot perform multiple commands in same block

```
%%sql
-- SQL Code Block
SELECT 'Hello' AS MSG
SELECT 'Goodbye' AS MSG
```

```
 * mysql://root:***@localhost/asteroids
(MySQLdb.ProgrammingError) (1064, "You have an err
server version for the right syntax to use near 'SE
[SQL: -- SQL Code Block
SELECT 'Hello' AS MSG
SELECT 'Goodbye' AS MSG]
(Background on this error at: https://sqlalche.me/
```

SQL Code Line

We can use our **Jupyter Lab** environment to execute **SQL code lines**.

```
%sql -- SQL Code Line
%sql SELECT 'Hello Gene' AS MSG
```

```
 * mysql://root:***@localhost/asteroids
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

MSG
Hello Gene

```
%sql -- SQL Code Line
%sql SELECT 'Hello Gene' AS MSG
%sql SELECT 'Hello IS-664' AS MSG
%sql SELECT CONCAT('Hello',' ','Everybody') AS MSG
```

```
 * mysql://root:***@localhost/asteroids
 * mysql://root:***@localhost/asteroids
1 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

MSG
Hello Everybody

Only last command is visible

SQL Code Line

We can assign **SQL code lines to variables** and display them

```
%sql -- SQL Code Line
A = %sql SELECT 'Hello Gene' AS MSG
B = %sql SELECT 'Hello IS-664' AS MSG

print(A)
print(B)
```

```
 * mysql://root:***@localhost/asteroids
 * mysql://root:***@localhost/asteroids
1 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
+------------+
|    MSG     |
+------------+
| Hello Gene |
+------------+
+--------------+
|     MSG      |
+--------------+
| Hello IS-664 |
+--------------+
```

```
%sql -- SQL Code Line
A = %sql SELECT JSON_ARRAY(1,2,3,4,5)
B = %sql SELECT JSON_ARRAY(8,3,4,5)
C = %sql SELECT JSON_LENGTH(JSON_ARRAY(1,2,3,4,5)) + JSON_LENGTH(JSON_ARRAY(8,3,4,5)) AS Array_Length

print(A)
print(B)
print(C)
```

```
 * mysql://root:***@localhost/asteroids
 * mysql://root:***@localhost/asteroids
1 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
+-----------------------+
| JSON_ARRAY(1,2,3,4,5) |
+-----------------------+
|    [1, 2, 3, 4, 5]    |
+-----------------------+
+---------------------+
| JSON_ARRAY(8,3,4,5) |
+---------------------+
|     [8, 3, 4, 5]    |
+---------------------+
+--------------+
| Array_Length |
+--------------+
|      9       |
+--------------+
```

SQL Code Line

We can assign **SQL code lines to session variables** and manipulate them

```
%sql -- SQL Code Line
%sql SET @A = 12
%sql SET @B = 10
%sql SELECT @A + @B AS Value
```

```
 * mysql://root:***@localhost/asteroids
 * mysql://root:***@localhost/asteroids
0 rows affected.
 * mysql://root:***@localhost/asteroids
0 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

Value
22

```
%sql -- SQL Code Line
%sql SET @A = 12
%sql SET @B = 10
%sql SELECT CONCAT_WS(' ',@A,'+',@B,'is',@A + @B) AS Value
```

```
 * mysql://root:***@localhost/asteroids
 * mysql://root:***@localhost/asteroids
0 rows affected.
 * mysql://root:***@localhost/asteroids
0 rows affected.
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

Value
12 + 10 is 22

Simple Queries

We can use our **Jupyter Lab** environment to **query our database** to examine results.

[30]:
```
%%sql
select * from spatialCoord limit 5;
```

```
 * mysql://root:***@localhost/asteroids
5 rows affected.
```

[30]:

Designation	X	Y	Z
C-a1872-l	3.39	4.93	4.57
C-a2151-m	4.63	4.20	3.19
C-a2440-j	3.70	4.52	3.40
C-a279-j	3.34	4.32	4.62
C-a39-l	3.23	3.43	4.46

[31]:
```
%%sql
SELECT R.Designation, R.Country, SC.X, SC.Y, SC.Z
FROM registry R
JOIN spatialCoord SC ON R.Designation = SC.Designation
LIMIT 5;
```

```
 * mysql://root:***@localhost/asteroids
5 rows affected.
```

[31]:

Designation	Country	X	Y	Z
C-a1872-l	US	3.39	4.93	4.57
C-a2151-m	UK	4.63	4.20	3.19
C-a2440-j	UK	3.70	4.52	3.40
C-a279-j	UK	3.34	4.32	4.62
C-a39-l	UK	3.23	3.43	4.46

MySQL Functions

We can use our **Jupyter Lab** environment to execute **native MySQL functions**.

```
%%sql
SELECT POW(10,2) AS Result
```

```
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

Result
100.0

```
%%sql
SELECT JSON_ARRAY(1,2,3,4,5)
```

```
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

JSON_ARRAY(1,2,3,4,5)
[1, 2, 3, 4, 5]

MySQL Functions

We can use our **Jupyter Lab** environment to execute **user-defined MySQL functions**.

```
%%sql
DROP FUNCTION IF EXISTS testme;
CREATE FUNCTION testme()
RETURNS VARCHAR(100)
DETERMINISTIC

BEGIN
    DECLARE A VARCHAR(100);
    SET A = 'HELLO JUPYTER';
    RETURN A;
END ;

SELECT testme() AS MSG;
```

```
 * mysql://root:***@localhost/asteroids
0 rows affected.
0 rows affected.
1 rows affected.
```

MSG
HELLO JUPYTER

```
%%sql
DROP FUNCTION IF EXISTS getCountry;
CREATE FUNCTION getCountry(DESG VARCHAR(20))
RETURNS VARCHAR(100)
NOT DETERMINISTIC
READS SQL DATA

BEGIN
    DECLARE A VARCHAR(100);
    SELECT Country FROM Registry WHERE Designation = DESG INTO A;
    RETURN A;
END ;

SELECT getCountry('C-a1872-l') AS MSG;
```

```
 * mysql://root:***@localhost/asteroids
0 rows affected.
0 rows affected.
1 rows affected.
```

MSG
US

MySQL Procedures

We can use our **Jupyter Lab** environment to execute **user-defined MySQL procedures**.

```
%%sql
DROP PROCEDURE IF EXISTS getCarbonAsteroids;
CREATE PROCEDURE getCarbonAsteroids(CNTRY VARCHAR(20))
BEGIN
    DECLARE I INT; DECLARE R INT;
    DECLARE DESG VARCHAR(20); DECLARE ATYP VARCHAR(20); DECLARE ADTE VARCHAR(20);
    DECLARE X CURSOR FOR
    SELECT Designation, AType, DDate
    FROM registry
    WHERE Country = CNTRY AND AType = 'Carboneous'
    LIMIT 5;

    SET I = 0;
    SET R = FOUND_ROWS();
    OPEN X;
    WHILE I < R DO
        FETCH X INTO DESG,ATYP,ADTE;
        SELECT CONCAT_WS(' ',DESG,ATYP,ADTE);
        SET I = I + 1;
    END WHILE;
    CLOSE X;
END;
```

```
 * mysql://root:***@localhost/asteroids
0 rows affected.
0 rows affected.
[]
```

```
%sql CALL getCarbonAsteroids('US');
```

```
 * mysql://root:***@localhost/asteroids
1 rows affected.
```

CONCAT_WS(' ',DESG,ATYP,ADTE)
C-a1872-I Carboneous 2007-09-02

Only last command is visible

MySQL Procedures

We can use our **Jupyter Lab** environment to execute **user-defined MySQL procedures**.

```
%%sql
DROP PROCEDURE IF EXISTS getCarbonAsteroids;
CREATE PROCEDURE getCarbonAsteroids(CNTRY VARCHAR(20))
BEGIN
    DECLARE I INT; DECLARE R INT;
    DECLARE DESG VARCHAR(20); DECLARE ATYP VARCHAR(20); DECLARE ADTE VARCHAR(20);
    DECLARE X CURSOR FOR
    SELECT Designation, AType, DDate
    FROM registry
    WHERE Country = CNTRY AND AType = 'Carboneous'
    LIMIT 5;

    DROP TABLE IF EXISTS carbons;
    CREATE TABLE carbons(
    A_DESG VARCHAR(20),
    A_ATYP VARCHAR(20),
    A_ADTE DATE,
    CONSTRAINT PK_Carbons PRIMARY KEY(A_DESG)
    );

    SET I = 0;
    OPEN X;
    SET R = FOUND_ROWS();
    WHILE I < R DO
        FETCH X INTO DESG,ATYP,ADTE;
        INSERT INTO carbons VALUES(DESG,ATYP,ADTE);
        SET I = I + 1;
    END WHILE;
    CLOSE X;
    SELECT * FROM carbons;
END;
```

```
 * mysql://root:***@localhost/asteroids
0 rows affected.
0 rows affected.
[]
```

```
    END WHILE;
    CLOSE X;
    SELECT * FROM carbons;
END;
```

Execute Last Command

```
%sql CALL getCarbonAsteroids('US');
```

```
 * mysql://root:***@localhost/asteroids
5 rows affected.
```

A_DESG	A_ATYP	A_ADTE
C-a1872-l	Carboneous	2007-09-02
C-e4604-p	Carboneous	2002-06-05
C-f2261-k	Carboneous	1996-01-23
C-f3770-k	Carboneous	2020-03-11
C-g1438-l	Carboneous	1993-03-11

MySQL Procedures

We can use our **Jupyter Lab** environment to execute **user-defined MySQL procedures**.

```
%%sql
DROP PROCEDURE IF EXISTS getCarbonAsteroids;
CREATE PROCEDURE getCarbonAsteroids(CNTRY VARCHAR(20))
BEGIN
    DECLARE I INT; DECLARE R INT;
    DECLARE DESG VARCHAR(20); DECLARE ATYP VARCHAR(20); DECLARE ADTE VARCHAR(20);
    DECLARE X CURSOR FOR
    SELECT Designation, AType, DDate
    FROM registry
    WHERE Country = CNTRY AND AType = 'Carboneous'
    LIMIT 5;

    DROP TABLE IF EXISTS carbons;
    CREATE TABLE carbons(
    A_DESG VARCHAR(20),
    A_ATYP VARCHAR(20),
    A_ADTE DATE,
    CONSTRAINT PK_Carbons PRIMARY KEY(A_DESG)
    );

    SET I = 0;
    OPEN X;
    SET R = FOUND_ROWS();
    WHILE I < R DO
        FETCH X INTO DESG,ATYP,ADTE;
        INSERT INTO carbons VALUES(DESG,ATYP,ADTE);
        SET I = I + 1;
    END WHILE;
    CLOSE X;
    SELECT * FROM carbons;
END;
```

```
 * mysql://root:***@localhost/asteroids
0 rows affected.
0 rows affected.
[]
```

```
L = %sql CALL getCarbonAsteroids('US');
print(L)
print()
print(L[0])
print()
print(L[0][1])
```

Use of Python List to hold values

```
 * mysql://root:***@localhost/asteroids
5 rows affected.
+-----------+------------+------------+
|   A_DESG  |   A_ATYP   |   A_ADTE   |
+-----------+------------+------------+
| C-a1872-l | Carboneous | 2007-09-02 |
| C-e4604-p | Carboneous | 2002-06-05 |
| C-f2261-k | Carboneous | 1996-01-23 |
| C-f3770-k | Carboneous | 2020-03-11 |
| C-g1438-l | Carboneous | 1993-03-11 |
+-----------+------------+------------+

('C-a1872-l', 'Carboneous', datetime.date(2007, 9, 2))

Carboneous
```

MySQL and Pandas

We can use **Pandas** (Python Library) in our **Jupyter Lab** environment to **read tables** into a manipulatable **data frame**.

```
%load_ext sql
connect = "mysql://root:root@localhost/asteroids"
%sql $connect
```

```
import pandas as pd
```

```
dfDB = pd.read_sql_table('registry',connect)
dfDB
```

	Designation	AType	Country	DDate
0	C-a1872-l	Carboneous	US	2007-09-02
1	C-a2151-m	Carboneous	UK	1994-08-08
2	C-a2440-j	Carboneous	UK	1991-10-27
3	C-a279-j	Carboneous	UK	2015-01-08
4	C-a39-l	Carboneous	UK	2013-08-19
...	...	...	...	...
95	S-h2054-k	Silicaceous	US	1994-07-15
96	S-h2242-q	Silicaceous	US	1999-05-08
97	S-h4510-j	Silicaceous	US	2009-11-19
98	S-h589-n	Silicaceous	CHINA	1997-06-05
99	S-h892-n	Silicaceous	RUSSIA	2014-12-24

100 rows × 4 columns

Pandas read_sql_table reads database tables into data frame.

MySQL and Pandas

We can use **Pandas** (Python Library) in our **Jupyter Lab** environment to **read tables** into a manipulatable **data frame**.

```
import pandas as pd
```

```
dfDB = pd.read_sql_table('registry',connect)
dfDB
```

	Designation	AType	Country	DDate
0	C-a1872-l	Carboneous	US	2007-09-02
1	C-a2151-m	Carboneous	UK	1994-08-08
2	C-a2440-j	Carboneous	UK	1991-10-27
3	C-a279-j	Carboneous	UK	2015-01-08
4	C-a39-l	Carboneous	UK	2013-08-19
...	...	...	...	...
95	S-h2054-k	Silicaceous	US	1994-07-15
96	S-h2242-q	Silicaceous	US	1999-05-08
97	S-h4510-j	Silicaceous	US	2009-11-19
98	S-h589-n	Silicaceous	CHINA	1997-06-05
99	S-h892-n	Silicaceous	RUSSIA	2014-12-24

100 rows × 4 columns

```
dfDB['Country']
```

```
0         US
1         UK
2         UK
3         UK
4         UK
       ...
95        US
96        US
97        US
98     CHINA
99    RUSSIA
Name: Country, Length: 100, dtype: object
```

We can reference individual columns

MySQL and Pandas

We can use **Pandas** (Python Library) in our **Jupyter Lab** environment to **capture data** from queries into a manipulatable **data frame**.

```
Q = "SELECT * FROM registry WHERE Country = 'US' LIMIT 5"
dfDB2 = pd.read_sql_query(Q,connect)
dfDB2
```

	Designation	AType	Country	DDate
0	C-a1872-l	Carboneous	US	2007-09-02
1	C-e4604-p	Carboneous	US	2002-06-05
2	C-f2261-k	Carboneous	US	1996-01-23
3	C-f3770-k	Carboneous	US	2020-03-11
4	C-g1438-l	Carboneous	US	1993-03-11

Pandas read_sql_query reads database query into data frame.

```
Q = "SELECT * FROM registry WHERE Country = 'US' LIMIT 5"
dfDB2 = pd.read_sql_query(Q,connect)
dfDB2['Designation'][0]
```

```
'C-a1872-l'
```

MySQL and Pandas

We can use **Pandas** (Python Library) in our **Jupyter Lab** environment to **capture data** from **functions** and **procedures** into a manipulatable **data frame**.

```
dfDB2 = pd.read_sql_query("SELECT getCountry('C-a1872-l') AS CNTRY",connect)
dfDB2
```

	CNTRY
0	US

We can call functions

```
dfDB2 = pd.read_sql_query("CALL getCarbonAsteroids('US')",connect)
dfDB2
```

	A_DESG	A_ATYP	A_ADTE
0	C-a1872-l	Carboneous	2007-09-02
1	C-e4604-p	Carboneous	2002-06-05
2	C-f2261-k	Carboneous	1996-01-23
3	C-f3770-k	Carboneous	2020-03-11
4	C-g1438-l	Carboneous	1993-03-11

We can call procedures

MySQL and Pandas

We can use **Pandas** (Python Library) in our **Jupyter Lab** environment to **capture data from SQL command files** into a manipulatable **data frame**.

test.sql | DB_Ana

```
CALL getCarbonAsteroids('US');
```

```
F = open('test.sql','r')
DF = pd.read_sql(F.read(),connect)
F.close()
DF
```

	A_DESG	A_ATYP	A_ADTE
0	C-a1872-l	Carboneous	2007-09-02
1	C-e4604-p	Carboneous	2002-06-05
2	C-f2261-k	Carboneous	1996-01-23
3	C-f3770-k	Carboneous	2020-03-11
4	C-g1438-l	Carboneous	1993-03-11

Pandas read_sql reads SQL script into data frame.

MySQL and NumPy

We can use **NumPy** (Python Library) in our **Jupyter Lab** environment to manipulate **data** from queries that have been stored in a **Pandas data frame**.

```
dfDB2 = pd.read_sql_query("SELECT * FROM specifications LIMIT 10",connect)
dfDB2
```

	Designation	Diameter	Mass	Density	Inclination	Rotation
0	C-a1872-l	630.428	106.925	1.046	26.325	12.74
1	C-a2151-m	694.952	689.171	1.199	28.102	11.50
2	C-a2440-j	69.375	265.537	1.743	27.437	24.09
3	C-a279-j	670.687	754.998	1.697	23.813	2.28
4	C-a39-l	846.326	272.581	1.417	12.220	14.99
5	C-b1038-p	634.667	301.133	1.954	14.085	17.25
6	C-b380-k	298.559	583.694	1.024	24.274	5.89
7	C-d5011-l	674.939	1040.617	1.497	12.243	12.52
8	C-e162-m	483.308	468.452	1.538	23.334	9.69
9	C-e1734-j	491.890	983.326	1.604	14.181	2.73

NumPy allows us to create arrays from rows returned by our queries

Row Name

```
import numpy as np

data = np.array(dfDB2.iloc[0])
for i in data:
    print(i,end=' ')
```

```
C-a1872-l 630.428 106.925 1.046 26.325 12.74
```

MySQL and NumPy

We can use **NumPy** (Python Library) in our **Jupyter Lab** environment to manipulate **data** from queries that have been stored in a **Pandas data frame**.

```
dfDB2 = pd.read_sql_query("SELECT * FROM specifications LIMIT 10",connect)
dfDB2
```

	Designation	Diameter	Mass	Density	Inclination	Rotation
0	C-a1872-l	630.428	106.925	1.046	26.325	12.74
1	C-a2151-m	694.952	689.171	1.199	28.102	11.50
2	C-a2440-j	69.375	265.537	1.743	27.437	24.09
3	C-a279-j	670.687	754.998	1.697	23.813	2.28
4	C-a39-l	846.326	272.581	1.417	12.220	14.99
5	C-b1038-p	634.667	301.133	1.954	14.085	17.25
6	C-b380-k	298.559	583.694	1.024	24.274	5.89
7	C-d5011-l	674.939	1040.617	1.497	12.243	12.52
8	C-e162-m	483.308	468.452	1.538	23.334	9.69
9	C-e1734-j	491.890	983.326	1.604	14.181	2.73

NumPy allows us to create arrays from rows of returned by our queries

Row index

```
data = np.array(dfDB2.loc[7])
print(data)
```

```
['C-d5011-l' 674.939 1040.617 1.497 12.243 12.52]
```

In this example, row index and row name are the same

MySQL and NumPy

We can use **NumPy** (Python Library) in our **Jupyter Lab** environment to manipulate **data** from queries that have been stored in a **Pandas data frame**.

```
dfDB2 = pd.read_sql_query("SELECT * FROM specifications LIMIT 10",connect)
dfDB2
```

	Designation	Diameter	Mass	Density	Inclination	Rotation
0	C-a1872-l	630.428	106.925	1.046	26.325	12.74
1	C-a2151-m	694.952	689.171	1.199	28.102	11.50
2	C-a2440-j	69.375	265.537	1.743	27.437	24.09
3	C-a279-j	670.687	754.998	1.697	23.813	2.28
4	C-a39-l	846.326	272.581	1.417	12.220	14.99
5	C-b1038-p	634.667	301.133	1.954	14.085	17.25
6	C-b380-k	298.559	583.694	1.024	24.274	5.89
7	C-d5011-l	674.939	1040.617	1.497	12.243	12.52
8	C-e162-m	483.308	468.452	1.538	23.334	9.69
9	C-e1734-j	491.890	983.326	1.604	14.181	2.73

```
data = np.array(dfDB2['Diameter'])
print(data)
print()
print(data.mean())
```

```
[630.428 694.952  69.375 670.687 846.326 634.667 298.559 674.939 483.308
 491.89 ]

549.5131
```