SQL concepts and queries using MySQL

In our previous lecture we looked installation of MySQL and queries to create the Students table and to insert data into it. Let’s continue where we stopped.

We will now insert a few more rows into the Students table.

```
INSERT INTO STUDENTS (FIRSTNAME, LASTNAME, AGE, MAJOR) VALUES ('FRED','KRAFT', 23, 'I.T.');
INSERT INTO STUDENTS (FIRSTNAME, LASTNAME, AGE, MAJOR) VALUES ('FRED','HEKT', 23, 'I.T.');
INSERT INTO STUDENTS (FIRSTNAME, LASTNAME, AGE, MAJOR) VALUES ('GENE','HEKT', 20, 'I.T.');
INSERT INTO STUDENTS (FIRSTNAME, LASTNAME, AGE) VALUES ('BOB','HEKT', 23);
```

To verify if the newly inserted data is present in the Students table.

```
SELECT * FROM STUDENTS;
```

NOTE: * retrieves all rows and all columns.

To select individual columns and not all the columns, the column names have to be specified in the query.

```
SELECT FIRSTNAME,LASTNAME,MAJOR FROM STUDENTS;
```

While working with tables that have considerable amount of data, it is always useful to know how much data is present in the table. The **COUNT()** function determines the number of rows in the table.

```
SELECT COUNT(*) FROM STUDENTS;
```

The Students table also has duplicate firstname entries. If unique firstnames are to be extracted, the **DISTINCT** statement does this for us.

```
SELECT DISTINCT FIRSTNAME FROM STUDENTS;
```

The select query we looked at above retrieves all the data. If there is a requirement to retrieve a subset of data we can filter our select query using the “where” statement.

SELECT * FROM STUDENTS WHERE FIRSTNAME = 'GENE';

To add multiple filters, we can use the SQL **AND** clause to concatenate filtering conditions.

SELECT * FROM STUDENTS WHERE FIRSTNAME = 'GENE' AND LASTNAME='HEKT';

Execute the SQL statement to select all values from the Students table.

SELECT * FROM STUDENTS;

Notice that Major for the row of data for Bob says **NULL.** Null indicates no data in that column for that row. We can select rows that have Null data by

SELECT * FROM STUDENTS WHERE MAJOR IS NULL;

It’s converse:

SELECT * FROM STUDENTS WHERE MAJOR IS NOT NULL;

Will extract all rows from Students where Major is not null.

Update

The SQL UPDATE command allows the user to Update on or more rows in a table. Let’s use the Update statement to update Bob’s major.

In MySQL in order to update or delete data we have to set the safe updates flag to 0.

SET SQL_SAFE_UPDATES = 0;

We can now update or delete data from our table.

UPDATE STUDENTS SET MAJOR = 'I.T.' WHERE FIRSTNAME = 'BOB';

Pattern Matching

We can retrieve data from any table based on a pattern. The % character acts as a wildcard.

SELECT * FROM STUDENTS WHERE FIRSTNAME = 'F%';

SELECT * FROM STUDENTS WHERE FIRSTNAME = '%E%';

Primary key and Foreign Key

A **primary key** consists of one or more columns where the data contained within these column/columns uniquely identify each row in the table.

Primary key column/columns can never have duplicates and it can never be NULL.

A **foreign key** is a column or a set of columns in table that refers to the **primary key** in another table.

To understand both of these concepts better, let's create two more tables.

```
CREATE TABLE COURSES(

COURSENUMBER INT,

COURSENAME TEXT);
```

Inserting data into our newly created courses table:

```
INSERT INTO COURSES(COURSENUMBER, COURSENAME) VALUES (101,'PHYSICS');

INSERT INTO COURSES(COURSENUMBER, COURSENAME) VALUES (102,'CHEMISTRY');
```

Creating an Enrollment table:

```
CREATE TABLE ENROLLMENT(

STUDENTID INT,

COURSENUMBER INT,

LOCATION TEXT);
```

Inserting data into our newly created enrollment table:

```
INSERT INTO ENROLLMENT(STUDENTID, COURSENUMBER, LOCATION) VALUES (1001, 101,
'B401');
```

```
INSERT INTO ENROLLMENT(STUDENTID, COURSENUMBER, LOCATION) VALUES (1006, 103,
'B401');

INSERT INTO ENROLLMENT(STUDENTID, COURSENUMBER, LOCATION) VALUES (1002, 102,
'2405');

INSERT INTO ENROLLMENT(STUDENTID, COURSENUMBER, LOCATION) VALUES (1003, 101,
'B401');
```

We have created three tables in our database. Students, Courses and Enrollment. Think of the Students table that stores all the students in a university. Think of the Courses table that stores all the courses available to take at the university.

The enrollment table should ideally contain student enrollment information about students who are present in the Students table and it should also contain courses that are only present in the Courses table.

If we examined the information at the enrollment table closely, these is a column named studentid. Ideally this information should also be present in the students table so that we can verify only students in the Students table are present in the enrollment table.

Using the ALTER TABLE SQL command, we can add a new column to the Students table.

```
ALTER TABLE STUDENTS ADD COLUMN STUDENTID INT;
```

Now execute

```
SELECT * FROM STUDENTS;
```

You will notice that the studentid column has now been created and has NULL values. Let’s add some data to this column.

```
UPDATE STUDENTS SET STUDENTID = 1001 WHERE FIRSTNAME = 'FRED' AND LASTNAME =
'RENK';

UPDATE STUDENTS SET STUDENTID = 1002 WHERE FIRSTNAME = 'FRED' AND LASTNAME =
'KRAFT';

UPDATE STUDENTS SET STUDENTID = 1003 WHERE FIRSTNAME = 'FRED' AND LASTNAME =
'HEKT';

UPDATE STUDENTS SET STUDENTID = 1004 WHERE FIRSTNAME = 'GENE' AND LASTNAME =
'HEKT';
```

UPDATE STUDENTS SET STUDENTID = 1005 WHERE FIRSTNAME = BOB AND LASTNAME = 'RENK';

We now have all the data we need. Let's implement Primary and Foreign key functionalities.

To reiterate, our goal is to keep our data clean and consistent and this means, only students present in the Students table and courses present in the Courses table should be in the enrollment table.

Execute the below SQL statement.

SELECT * FROM ENROLLMENT;

Notice the second row that contains studentid 1006 not present in our students table. We should be able to avoid such errors in data. To accomplish this we have to create a Primary Key, Foreign Key relationship between Enrollment, Students and Courses.

Step 1) Create a Primary Key for the Student table. The student table has a studentid column that will be unique to every row so let's use this column as our primary key. Since the table has already been created, we use the Alter Table statement to add primary key.

ALTER TABLE STUDENTS ADD PRIMARY KEY(STUDENTID);

Step 2) Create a Primary Key for the Courses table. Courses table has the coursenumber column which will be unique for every row of data. Let's make this the Primary Key for the Courses table.

ALTER TABLE COURSES ADD PRIMARY KEY(COURSENUMBER);

Enrollment has columns studentid and coursenumber. These will be Foreign Keys that refer to studentid in Students and coursenumber in Courses. Using the Alter Table statement to add the Foreign Keys.

ALTER TABLE ENROLLMENT ADD FOREIGN KEY(STUDENTID)

REFERENCES STUDENTS(STUDENTID);

ALTER TABLE ENROLLMENT ADD FOREIGN KEY(COURSENUMBER)

REFERENCES COURSES(COURSENUMBER);

On execution of both the above queries, you will notice errors saying the Foreign Key constraint is violated.

This is because as things stand, the Enrollment table has data which is not consistent with both the Students and Courses table. Hence we have to clean our data before creating Foreign Keys.

Data cleaning can be done by either inserting the student(1006) not in the Students table or deleting this entry from the Enrollment table.

Let’s insert the data into Students.

INSERT INTO STUDENTS(STUDENTID, FIRSTNAME, LASTNAME, AGE, MAJOR) VALUES (1006, 'TED','HEKT',22,'MARKETING');

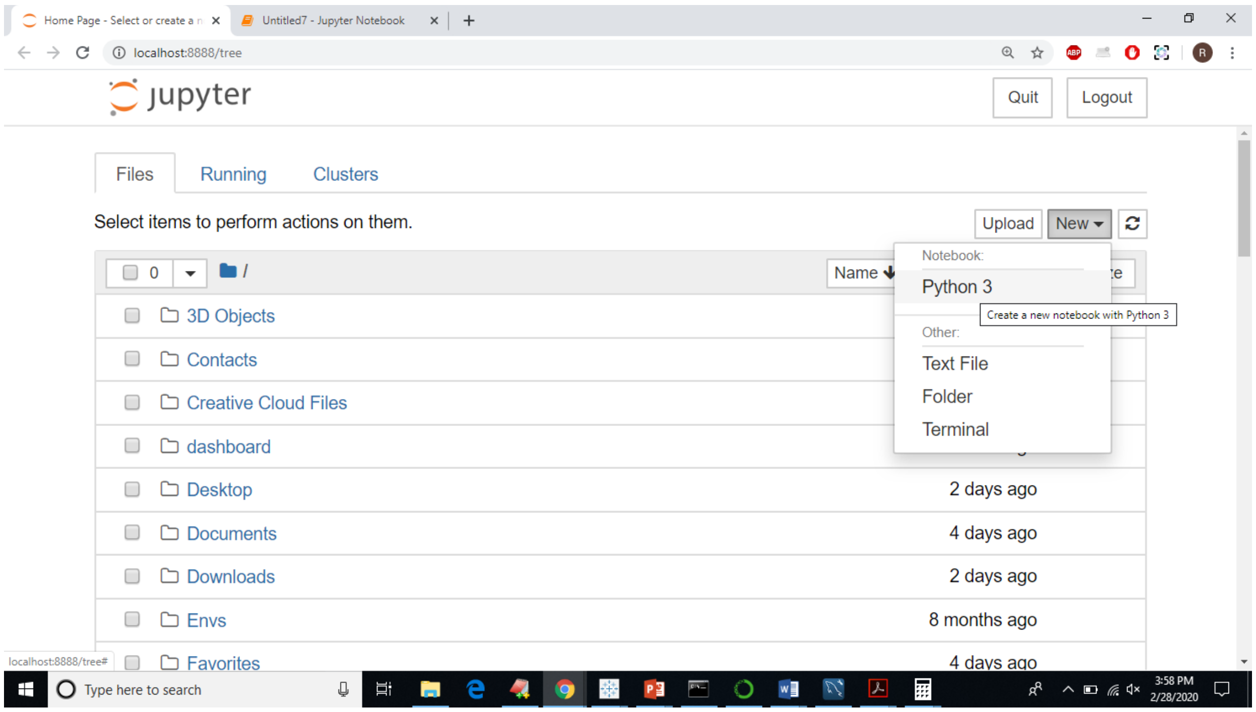
We have now cleaned up student information, all that’s left to do is to clean up the coursenumber in the Enrollment table that contain coursenumber 103 not present in the main Courses table.

UPDATE ENROLLMENT SET COURSENUMBER = 102 WHERE STUDENTID = 1006;

Now that the data is clean and consistent, go back and execute the Foreign Key creation commands.

Python and MySQL

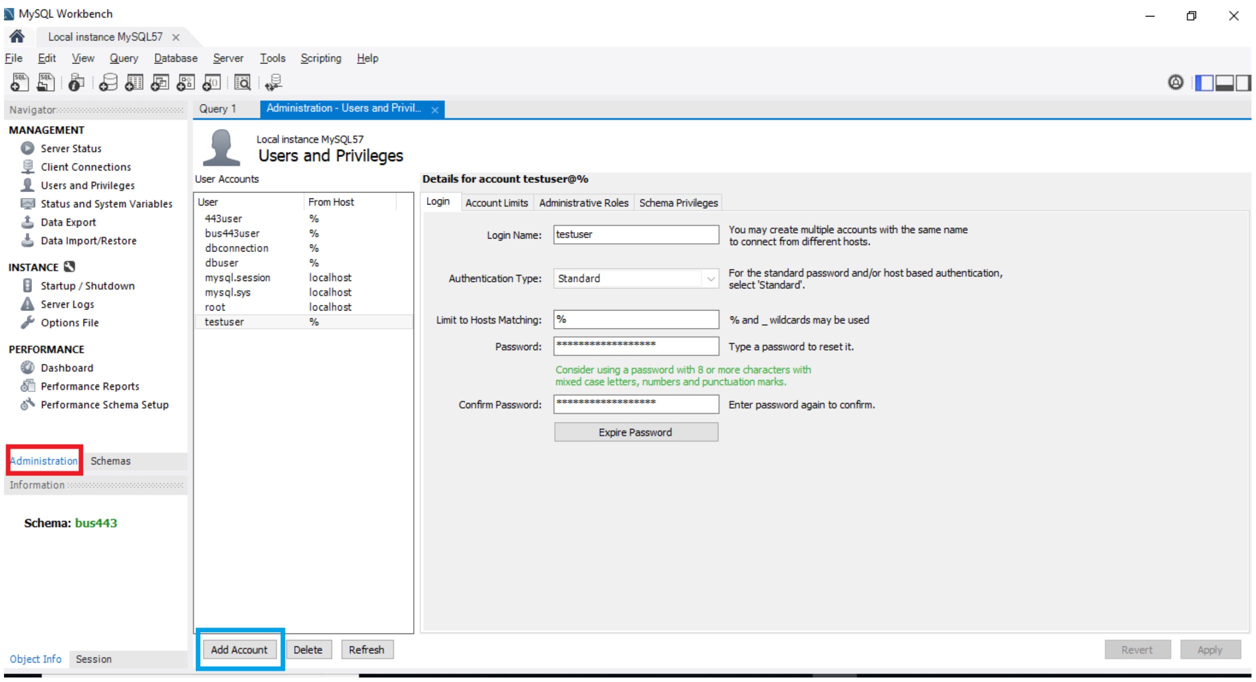
MySQL will be our database when we start creating web application in Django (python based web framework). Hence we have to connect python and MySQL. To see how this works, create a new Python 3 workbook in Jupyter Notebook.



Select items to perform actions on them.

3D Objects	
Contacts	
Creative Cloud Files	
dashboard	
Desktop	2 days ago
Documents	4 days ago
Downloads	2 days ago
Envs	8 months ago
Favorites	4 days ago

Before we begin with writing code to connect to MySQL, we should also create a new database user. To do this, open MySQL Workbench and click on Administration (Red below)



Schema: bus443

User Accounts

User	From Host
443user	%
bus443user	%
dbconnection	%
dbuser	%
mysql.session	localhost
mysql.sys	localhost
root	localhost
testuser	%

Details for account testuser@%

Login Name: testuser — You may create multiple accounts with the same name to connect from different hosts.

Authentication Type: Standard — For the standard password and/or host based authentication, select 'Standard'.

Limit to Hosts Matching: % — % and _ wildcards may be used

Password: ****************** — Type a password to reset it.

Consider using a password with 8 or more characters with mixed case letters, numbers and punctuation marks.

Confirm Password: ****************** — Enter password again to confirm.

Then click on Add Account (Blue above).

Provide a username and password and click apply. Remember the username and password that you enter. We will need this to connect to our database via python.

Next step is to import the python package that allows us to connect to MySQL.

In the code block in the Python 3 workbook. Execute the below commands.

```
from pip._internal import main

main(['install','mysql-connector-python-rf'])
```

Once this is successful, we can use our newly created username and password to connect to the database.

Execute the below commands in a new block.

```
import mysql.connector as mysql

db = mysql.connect(host="localhost", user="testuser", passwd="P@ssword")

cursor = db.cursor()

cursor.execute("SELECT * FROM BUS443.STUDENTS")

data = cursor.fetchall()

print(data)
```

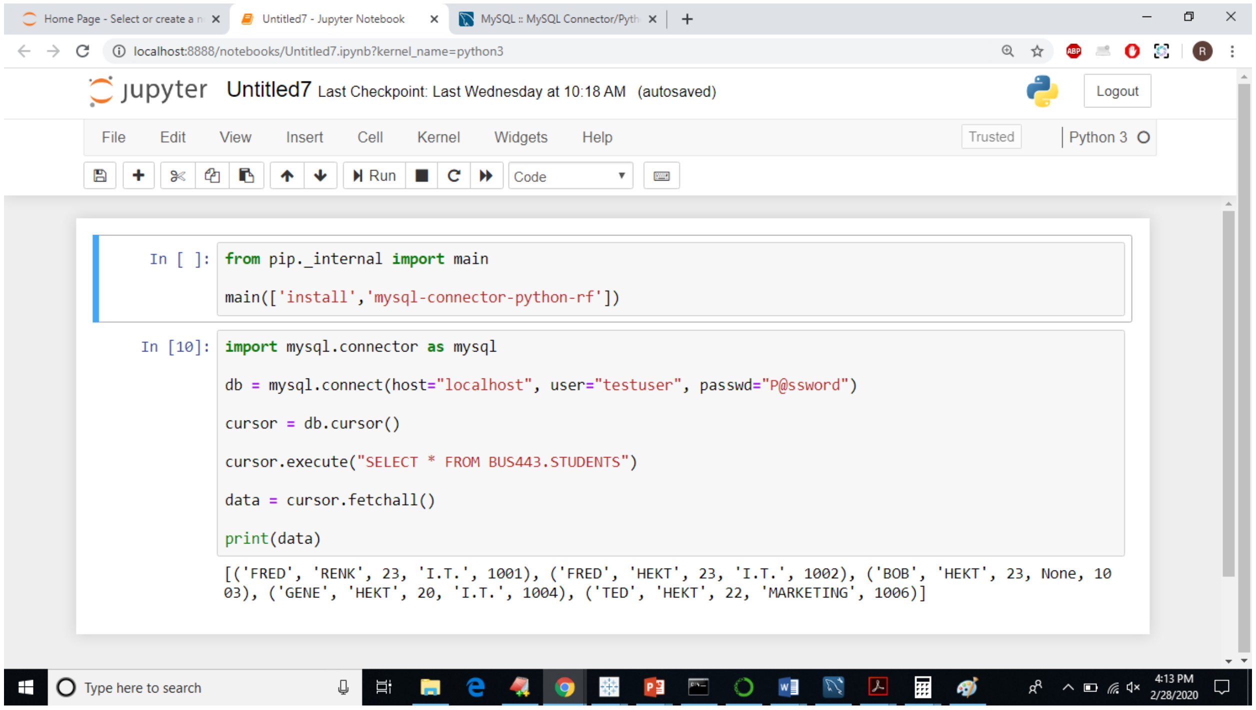
The first line, imports the package to the current working environment. While working with Python not all modules are available to you immediately. The modules or packages that are not present can be downloaded and imported using the *import* command.

import mysql.connector as mysql (***mysql*** here is an alias to shorten the name of the package once imported to the current environment)

The second line opens up a connection to the database and creates a database connection object.

Line 3 creates a new cursor object. Cursors allow us to execute database queries and to retrieve data from the database.

Line 4 executes the SQL statement and Line 5 retrieves all the results from the cursor object.



```
In [ ]: from pip._internal import main

        main(['install','mysql-connector-python-rf'])
```

```
In [10]: import mysql.connector as mysql

         db = mysql.connect(host="localhost", user="testuser", passwd="P@ssword")

         cursor = db.cursor()

         cursor.execute("SELECT * FROM BUS443.STUDENTS")

         data = cursor.fetchall()

         print(data)

         [('FRED', 'RENK', 23, 'I.T.', 1001), ('FRED', 'HEKT', 23, 'I.T.', 1002), ('BOB', 'HEKT', 23, None, 10
         03), ('GENE', 'HEKT', 20, 'I.T.', 1004), ('TED', 'HEKT', 22, 'MARKETING', 1006)]
```

Data is returned as a python list which contains a Tuple for each row of data.

TO DO: Look at more cursor examples. If you have questions about SQL, please look at tutorials online or visit me during office hours.