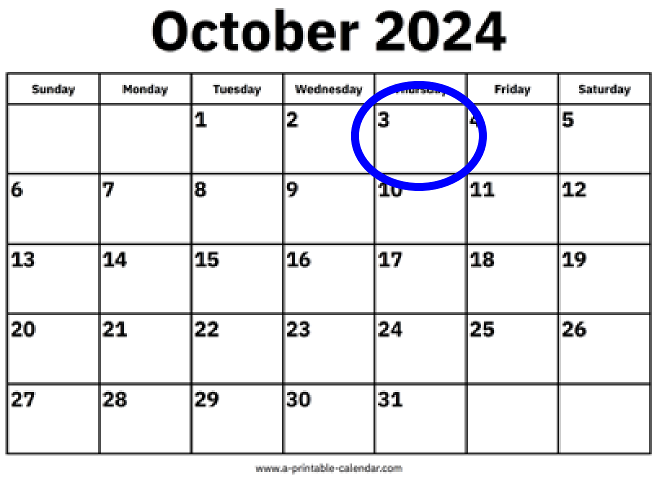
CS 310 : Scalable Software Architectures

Class session on Thursday, October 3rd



October 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

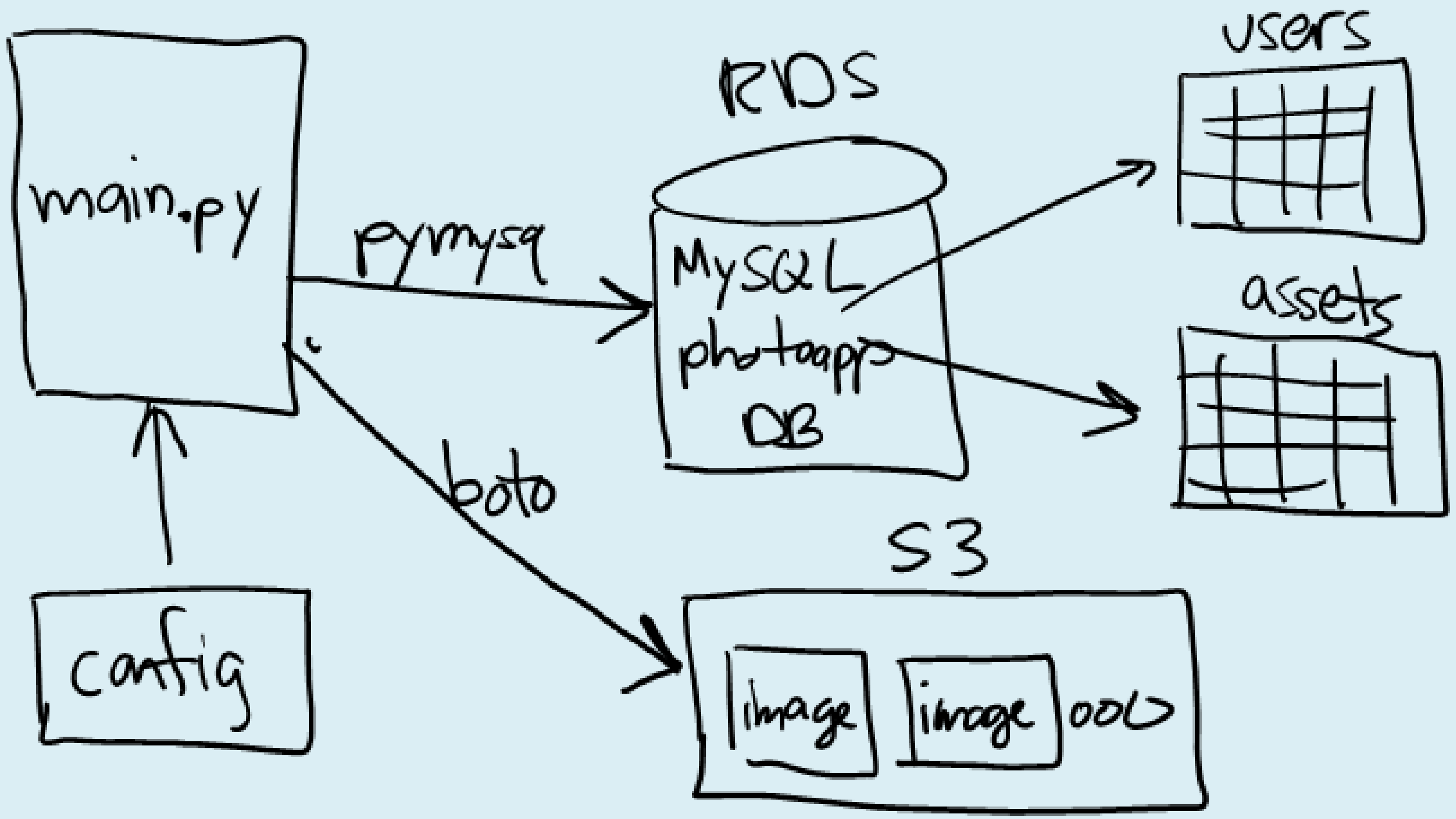
www.a-printable-calendar.com

Notes:

- ***Focus this week:***
  - *Relational databases*
- ***Class sessions *are* being recorded this week***
  - *Will be available under Panopto on Canvas*
- ***Project 01** due next Wednesday @ 11:59pm*
  - *Build a simple photo app using AWS*
  - *Parts 01 and 02 are both released*
  - *Can submit as late as Friday @ 11:59pm*
- ***Office hours** started Monday*
- *Optional SQL **homework** posted*

Northwestern University

Project 01

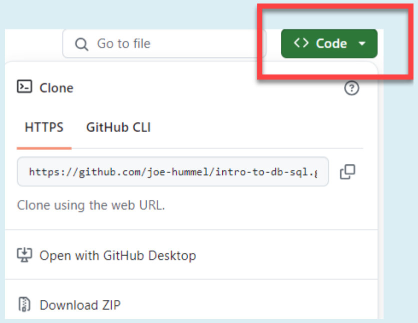

main.py
pymysq
RDS
MySQL
photoapp
DB
users
assets
boto
S3
image
image
ooo
config

Goals for today

- **Executing SQL from Python**
  - *With sqlite3, a local file-based DBMS*
  - *With MySQL running on a server in AWS*

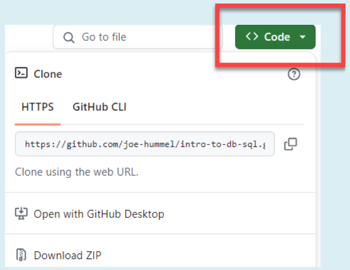
- **Work with Docker**
  - *Docker Desktop must be installed*
  - *Download files from GitHub:*
    - https://github.com/joe-hummel/intro-to-db-sql
    - Clone repo or download ZIP

Same files as Tuesday, no need to clone/download again



Getting the necessary software

1. **Download files you need for today**
   - https://github.com/joe-hummel/intro-to-db-sql
   - Clone repo or download ZIP



```
https://github.com/joe-hummel/intro-to-db-sql.
```

Clone using the web URL.

2. **Make sure Docker Desktop is running**

3. **Build Docker image and run container:**

Linux/Mac/Windows WSL:

1) Open terminal, navigate to repo folder
2) `chmod 755 *.bash`
3) `./docker-build.bash`
4) `./docker-run.bash`

Windows:

1) Open Powershell, navigate to repo folder
2) `.\docker-build.bat`
3) `.\docker-run.bat`

```
hummel> ./docker-run.bash
docker> ls
Dockerfile          datatier.py         docker-build.bat    docker-run.bat      main.sql
cta.db              docker-build.bash   docker-run.bash     main.py             movielens.db
docker>
```

Common docker errors

1. **"docker" command not found**
   - *Uninstall and reinstall Docker Desktop*

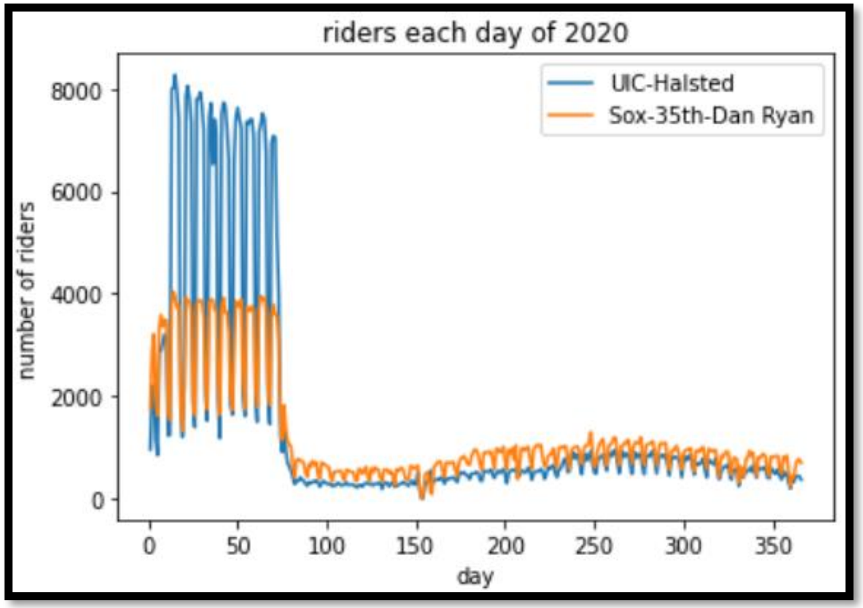
2. **When you try to build, you are not authorized**
   - *`docker login -u docker-username`*

3. **When you try to run, you get errors like "bash: $\r: command not found"**
   1. *If you see the **docker>** prompt, type **exit***
   2.
```
((Get-Content .bashrc) -join "`n") + "`n" | Set-Content -NoNewLine .bashrc
```

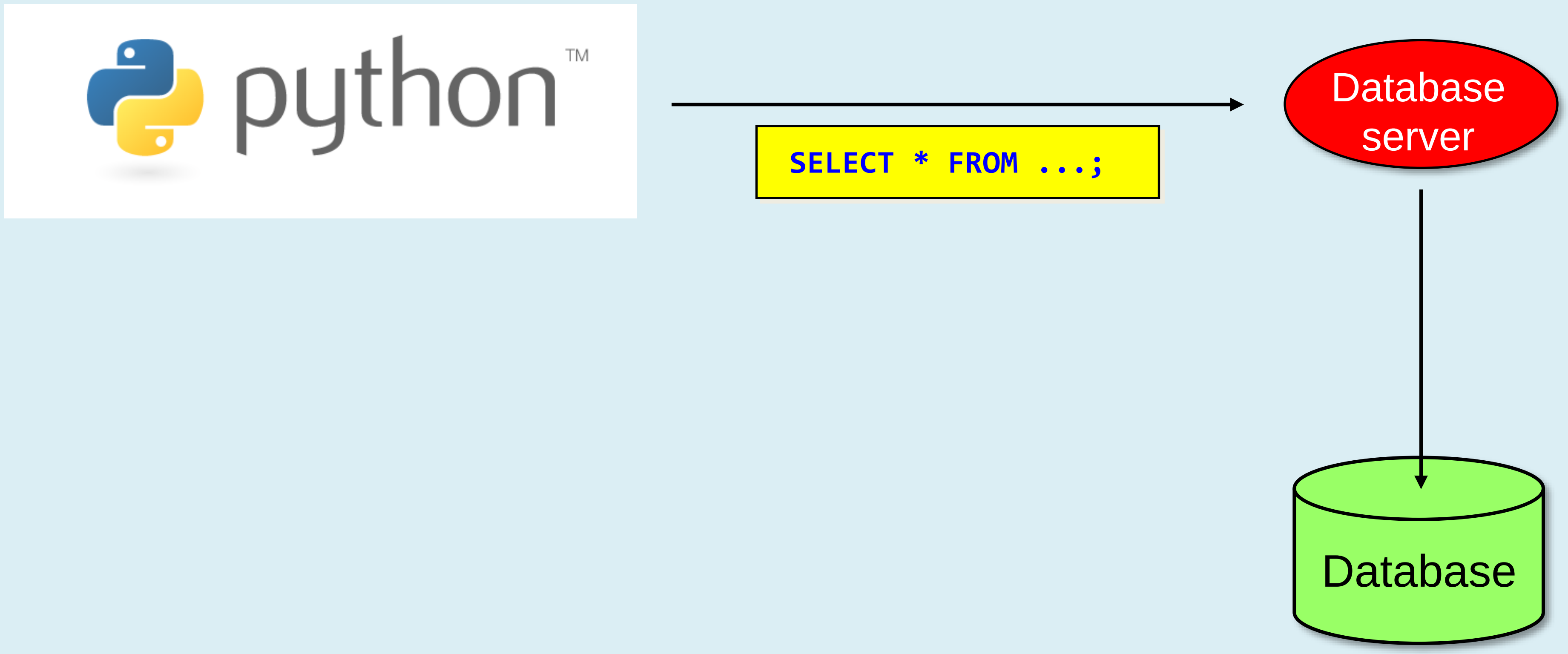
Executing SQL within other languages

- **SQL is a powerful language, but lacking in other areas**
  - *No UI support, no graphics, no web, no mobile*

- **SQL is commonly embedded within other languages**
  - *JavaScript*
  - *Python*
  - *Java*
  - *Swift*
  - *C#*
  - *etc.*



Executing SQL from Python



sqlite3 Python library

- **Open connection**

Cursor is a pointer to the data, and is used to scroll through the data

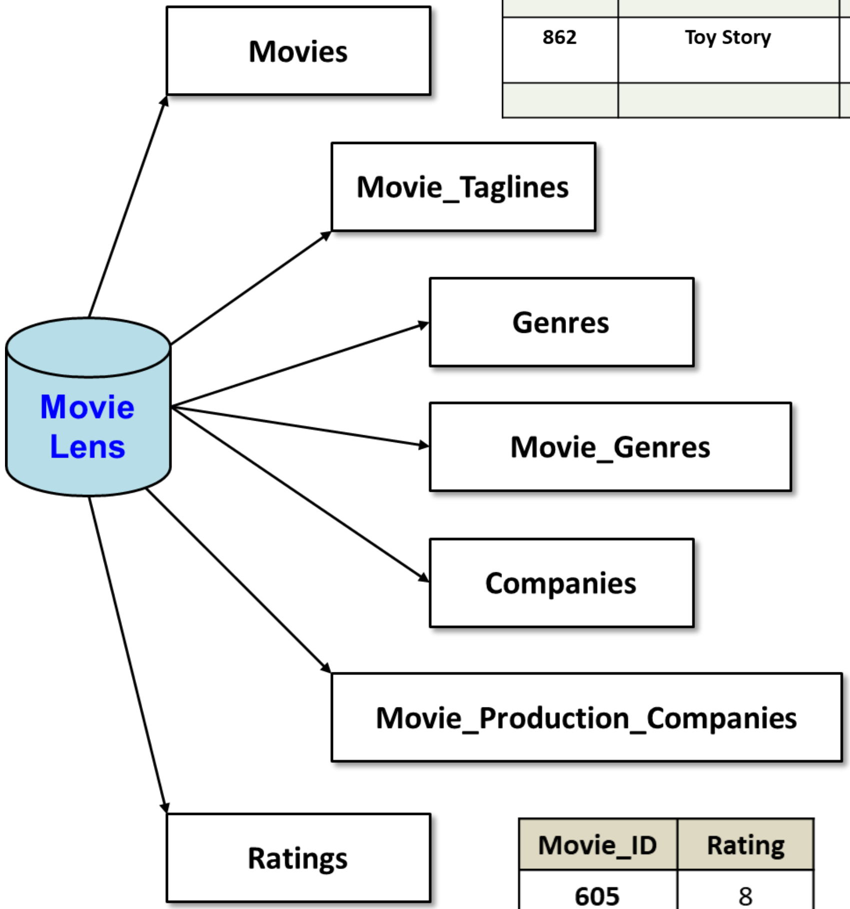
- **Create a cursor**
- **Execute SQL**
- **Fetch result**
  - *fetchone( )*
    - Returns a tuple (...)
  - *fetchall( )*
    - Returns a list of tuples

```
import sqlite3

dbConn = sqlite3.connect("filename.db")
dbCursor = dbConn.cursor()

# query to retrieve 1 row from DB:
sql = """
      Select …
      From …
      Where … ;
      """

dbCursor.execute(sql)
row = dbCursor.fetchone()

print(row)
```

```
# query to retrieve N values from DB:
sql = """
      Select …
      From …
      Where … ;
      """

dbCursor.execute(sql)
rows = dbCursor.fetchall()

for row in rows:
  print(row)
```

MovieLens database

Movie_ID	Title	Release_Date	Runtime	Original_L anguage	Budget	Revenue
603	**The Matrix**	1999-03-30 00:00:00.000	136	en	63000000	463517383
862	**Toy Story**	1995-10-30 00:00:00.000	81	en	30000000	373554033

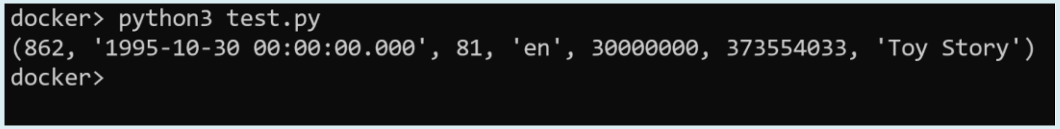


Movie_ID	Rating
605	8
603	6
605	10
605	6

Exercise #1

1. **Build docker image (if you haven't already)**
2. **Run docker image**
3. **Create a file in the repo folder named "test.py"**
4. **Open "test.py" in your favorite text editor**
5. **Code in Python**
   1) *Open MovieLens.db*
   2) *Retrieve all columns for 'Toy Story'*
   3) *Print the row*
6. **Docker>** `python3 test.py`

```
import sqlite3

dbConn = sqlite3.connect("movielens.db")
dbCursor = dbConn.cursor()

sql = """
      select * from movies
      where title = 'Toy Story';
      """

dbCursor.execute(sql)
row = dbCursor.fetchone()

print(row)
```



```
docker> python3 test.py
(862, '1995-10-30 00:00:00.000', 81, 'en', 30000000, 373554033, 'Toy Story')
docker>
```

Your turn

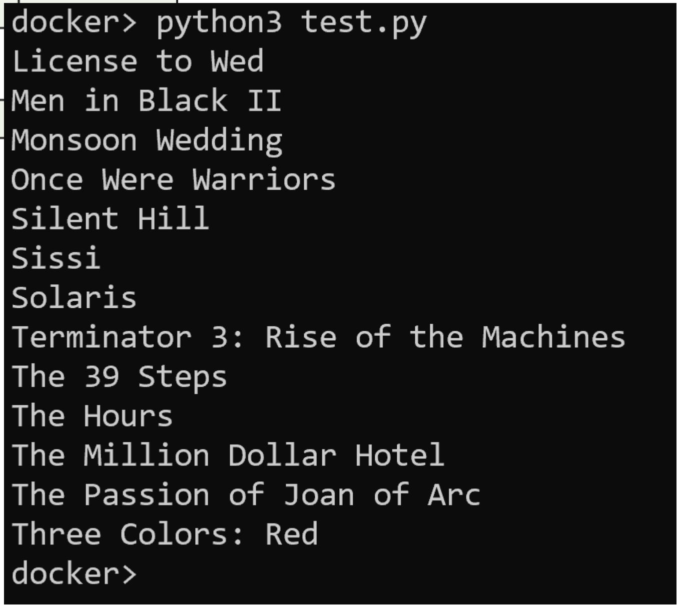
- **Output all movies that have > 200 reviews**
  - *You need a join, group by, and having*
  - *Use fetchall( )*

Movies

Movie_ID	Title	Release_Date	Runtime	Original_Language	Budget	Revenue
603	The Matrix	1999-03-30 00:00:00.000	136	en	63000000	463517383
862	Toy Story	1995-10-30 00:00:00.000	81	en	30000000	

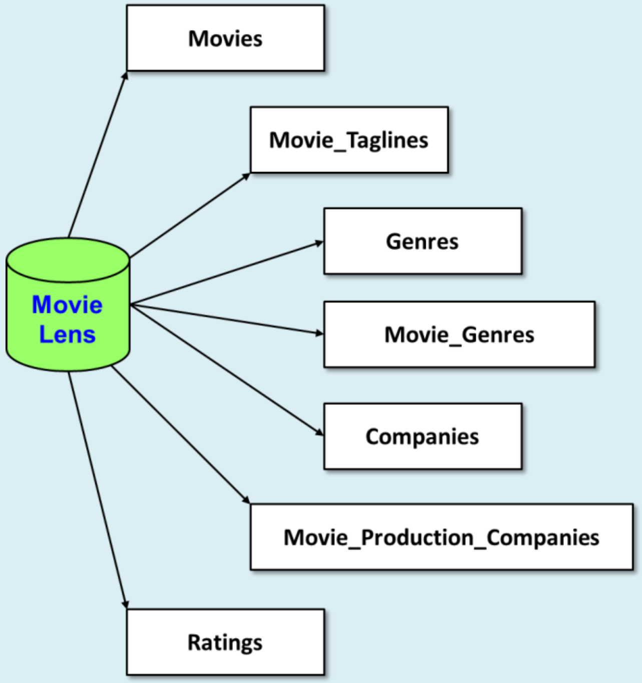
Ratings

Movie_ID	Rating
605	8
603	6
605	10
605	6

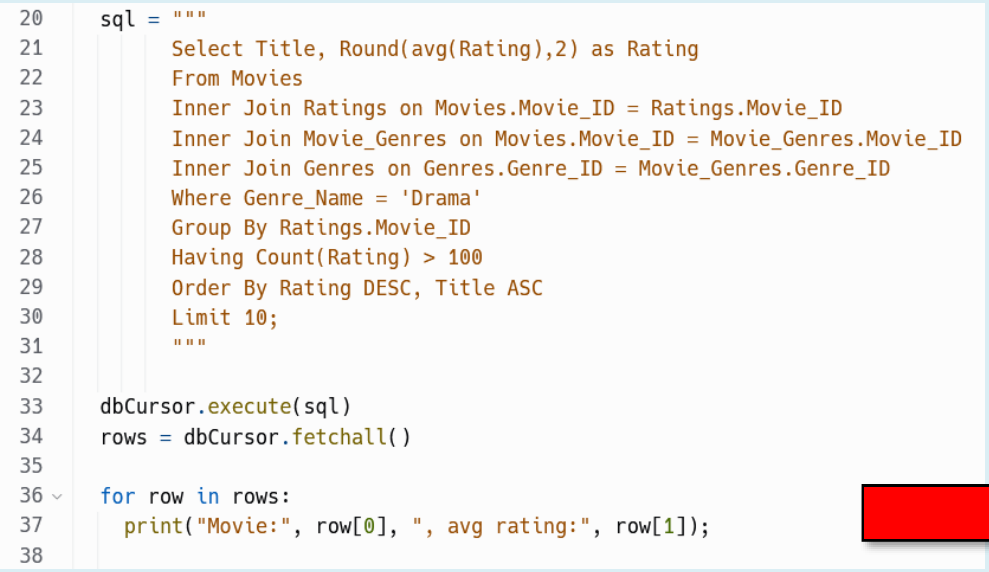


```
docker> python3 test.py
License to Wed
Men in Black II
Monsoon Wedding
Once Were Warriors
Silent Hill
Sissi
Solaris
Terminator 3: Rise of the Machines
The 39 Steps
The Hours
The Million Dollar Hotel
The Passion of Joan of Arc
Three Colors: Red
docker>
```

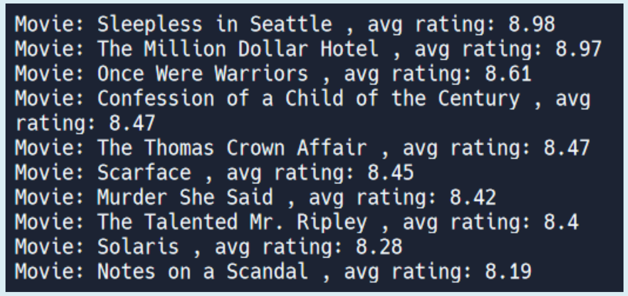
Top-10 movies in Drama genre



Top-10 Drama movies



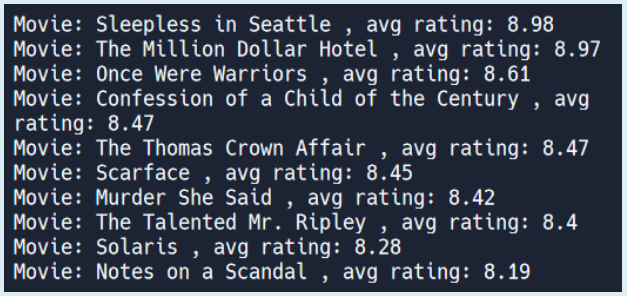
```python
sql = """
        Select Title, Round(avg(Rating),2) as Rating
        From Movies
        Inner Join Ratings on Movies.Movie_ID = Ratings.Movie_ID
        Inner Join Movie_Genres on Movies.Movie_ID = Movie_Genres.Movie_ID
        Inner Join Genres on Genres.Genre_ID = Movie_Genres.Genre_ID
        Where Genre_Name = 'Drama'
        Group By Ratings.Movie_ID
        Having Count(Rating) > 100
        Order By Rating DESC, Title ASC
        Limit 10;
        """

dbCursor.execute(sql)
rows = dbCursor.fetchall()

for row in rows:
  print("Movie:", row[0], ", avg rating:", row[1]);
```



```
Movie: Sleepless in Seattle , avg rating: 8.98
Movie: The Million Dollar Hotel , avg rating: 8.97
Movie: Once Were Warriors , avg rating: 8.61
Movie: Confession of a Child of the Century , avg
rating: 8.47
Movie: The Thomas Crown Affair , avg rating: 8.47
Movie: Scarface , avg rating: 8.45
Movie: Murder She Said , avg rating: 8.42
Movie: The Talented Mr. Ripley , avg rating: 8.4
Movie: Solaris , avg rating: 8.28
Movie: Notes on a Scandal , avg rating: 8.19
```

Execute the query

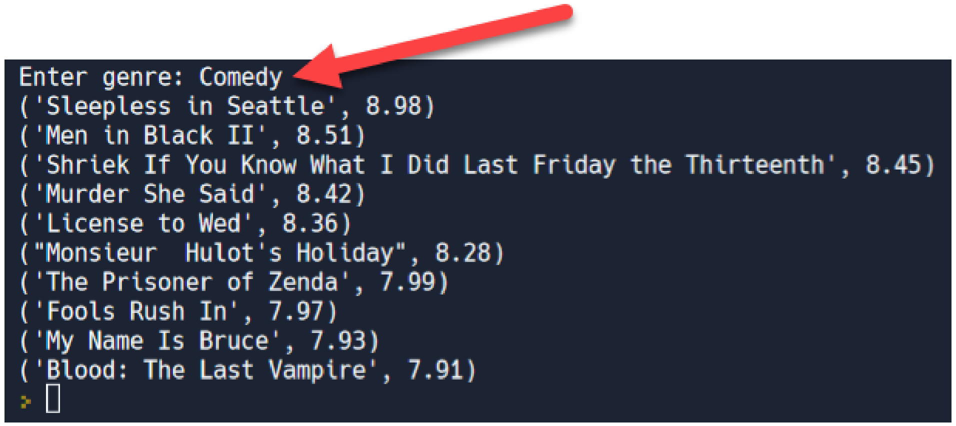
1. **The code is already provided in "main.py"**
2. **`Docker>`** `python3 main.py`



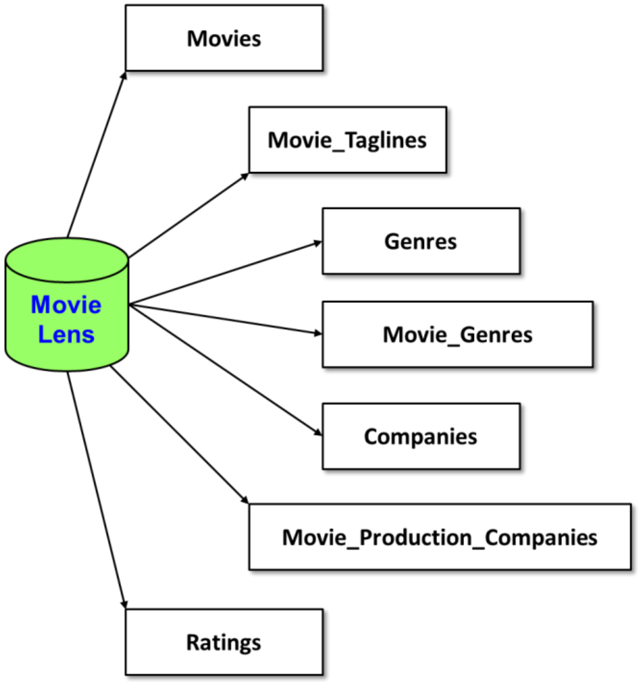
```
Movie: Sleepless in Seattle , avg rating: 8.98
Movie: The Million Dollar Hotel , avg rating: 8.97
Movie: Once Were Warriors , avg rating: 8.61
Movie: Confession of a Child of the Century , avg
rating: 8.47
Movie: The Thomas Crown Affair , avg rating: 8.47
Movie: Scarface , avg rating: 8.45
Movie: Murder She Said , avg rating: 8.42
Movie: The Talented Mr. Ripley , avg rating: 8.4
Movie: Solaris , avg rating: 8.28
Movie: Notes on a Scandal , avg rating: 8.19
```

Parameterized ("dynamic") queries

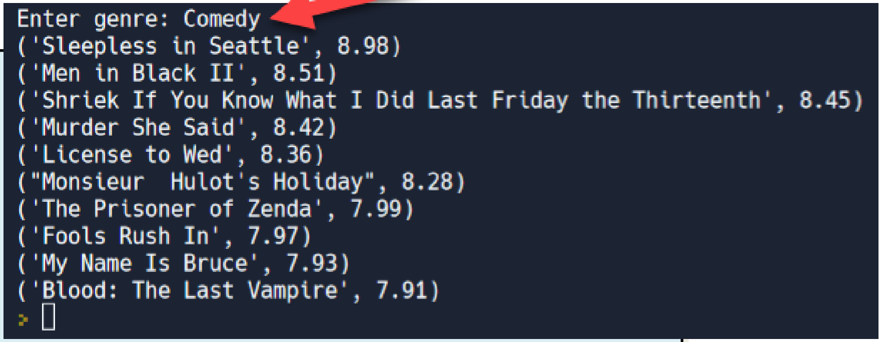
- **Most queries are dynamic, responding to user input**



```
Enter genre: Comedy
('Sleepless in Seattle', 8.98)
('Men in Black II', 8.51)
('Shriek If You Know What I Did Last Friday the Thirteenth', 8.45)
('Murder She Said', 8.42)
('License to Wed', 8.36)
("Monsieur  Hulot's Holiday", 8.28)
('The Prisoner of Zenda', 7.99)
('Fools Rush In', 7.97)
('My Name Is Bruce', 7.93)
('Blood: The Last Vampire', 7.91)
>
```



Executing dynamic queries



```
Enter genre: Comedy
('Sleepless in Seattle', 8.98)
('Men in Black II', 8.51)
('Shriek If You Know What I Did Last Friday the Thirteenth', 8.45)
('Murder She Said', 8.42)
('License to Wed', 8.36)
("Monsieur  Hulot's Holiday", 8.28)
('The Prisoner of Zenda', 7.99)
('Fools Rush In', 7.97)
('My Name Is Bruce', 7.93)
('Blood: The Last Vampire', 7.91)
>
```

```
import sqlite3

dbConn   = sqlite3.connect("chicago-police-stops.db")
dbCursor = dbConn.cursor()

genre = input("Enter genre: ")

sql = """
      Select Title, Round(avg(Rating),2) as Rating
      From   Movies
      Inner Join Ratings on Movies.Movie_ID = Ratings.Movie_ID
      Inner Join Movie_Genres on Movies.Movie_ID = Movie_Genres.Movie_ID
      Inner Join Genres on Genres.Genre_ID = Movie_Genres.Genre_ID
      Where Genre_Name = ?
      Group By Movies.Movie_ID
      Having Count(Rating) > 100
      Order By Rating DESC, Title ASC
      Limit 10;
      """

dbCursor.execute(sql, [genre])
rows = dbCursor.fetchall()

for row in rows:
  print(row)
```

placeholder…

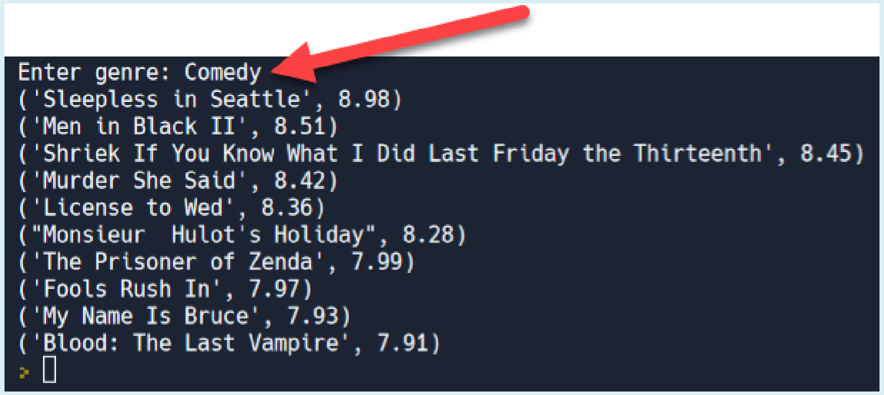
NOTE: with MySQL use %s instead of ?

Provide parameter value(s) in call so DBMS builds query and checks for injection attacks

Try it

1. **Modify "main.py" in your favorite editor**
   1) *Modify SQL query string, replacing 'Drama' with* ***?***
   2) *Pass genre value in call to execute(sql,* ***[genre]*** *)*
   3) *Run and try different genres (Comedy, Horror, …)*

2. **Docker>** `python3 main.py`



```
Enter genre: Comedy
('Sleepless in Seattle', 8.98)
('Men in Black II', 8.51)
('Shriek If You Know What I Did Last Friday the Thirteenth', 8.45)
('Murder She Said', 8.42)
('License to Wed', 8.36)
("Monsieur  Hulot's Holiday", 8.28)
('The Prisoner of Zenda', 7.99)
('Fools Rush In', 7.97)
('My Name Is Bruce', 7.93)
('Blood: The Last Vampire', 7.91)
>
```

Beware SQL injection attacks

```
genre = input("Enter genre: ")

sql = "…" + genre + "…"
sql = f"… {genre} …"

dbCursor.execute(sql)
rows = dbCursor.fetchall()

for row in rows:
  print(row)
```

NEVER build sql query strings yourself --- you open the door to SQL injection attacks

Attacks can delete your tables.

Attack Looks like this

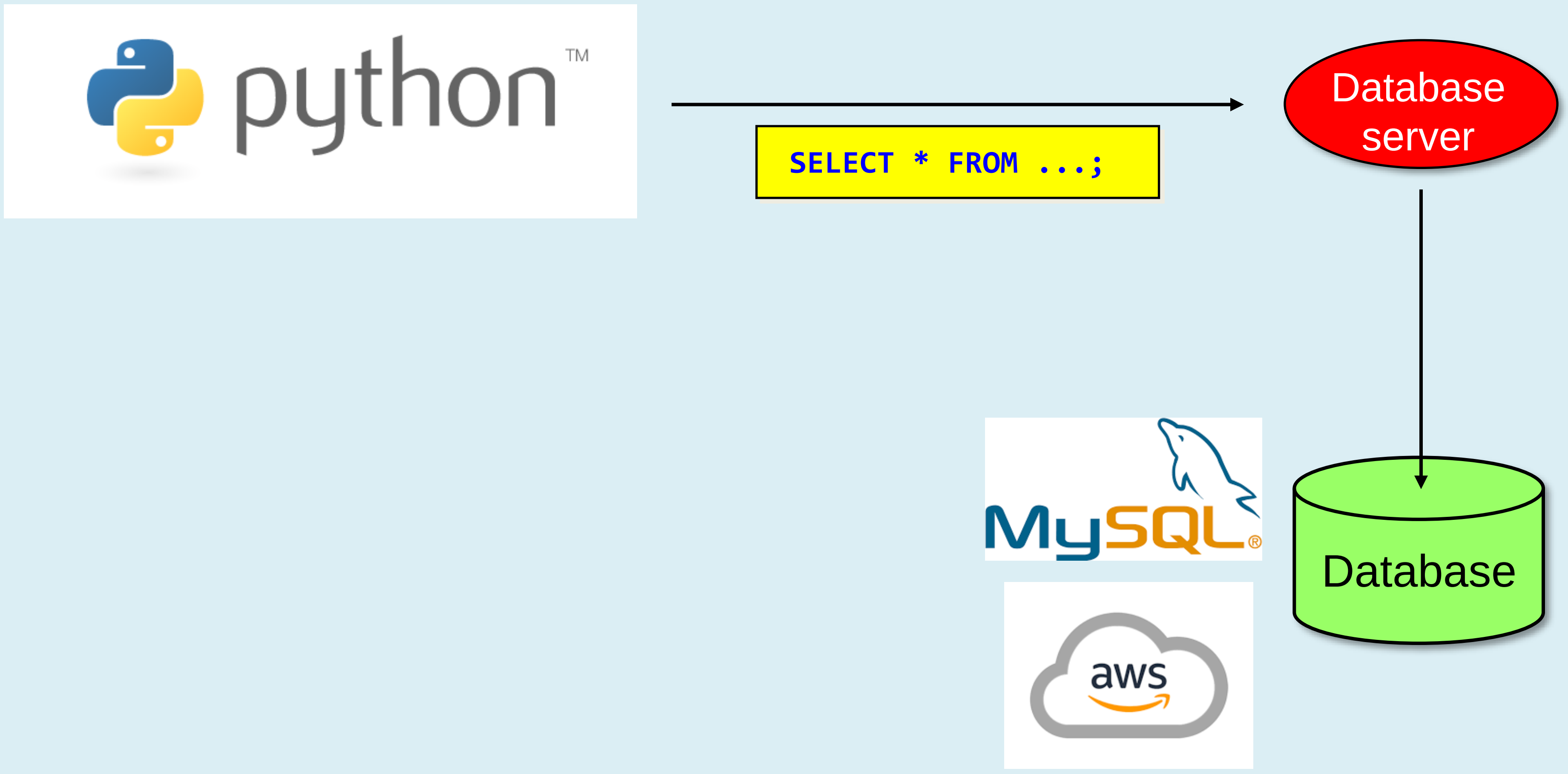
ignore'; delete from ratings; —

ignore will ignore all the above
delete will remove all the data
— comments out everything after

These are very common attacks. So never compile SQL with f strings

https://www.bleepingcomputer.com/news/security/researchers-find-sql-injection-to-bypass-airport-tsa-security-checks/

How about executing SQL with MySQL?



```
SELECT * FROM ...;
```

pymysql

- **Use pymysql module --- nearly identical API !**
- **https://pymysql.readthedocs.io/en/latest/**

```
import pymysql

try:
  dbconn =  pymysql.connect(
    host=ENDPOINT,
    port=PORTNUM,
    user=USERNAME,
    passwd=PASSWORD,
    database=DBNAME)

  sql = "…"

  dbCursor = dbconn.cursor()
  dbCursor.execute(sql)
  rows = dbCursor.fetchall()    # or fetchone()
  for row in rows:
    print(row)

except Exception as e:
  print("Database connection failed due to {}".format(e))
```

For dynamic queries use %s instead of ?

Try it

1. **Modify "Dockerfile" in your favorite editor**
   1) *On the last line add:* `RUN pip3 install pymysql`
   2) *Exit your docker container:* `exit`
   3) *Build docker image*
   4) *Run docker image*

2. **Download "main-mysql.py"**
   1) *https://tinyurl.com/main-mysql*
   2) *Move "main-mysql.py" into your repo folder*

3. **Docker>** `python3 main-mysql.py`

Try it with these genres:
Music
Family
Horror

That's it, thank you!