Programmatic Execution of SQL

This pdf is completed and recording is watched

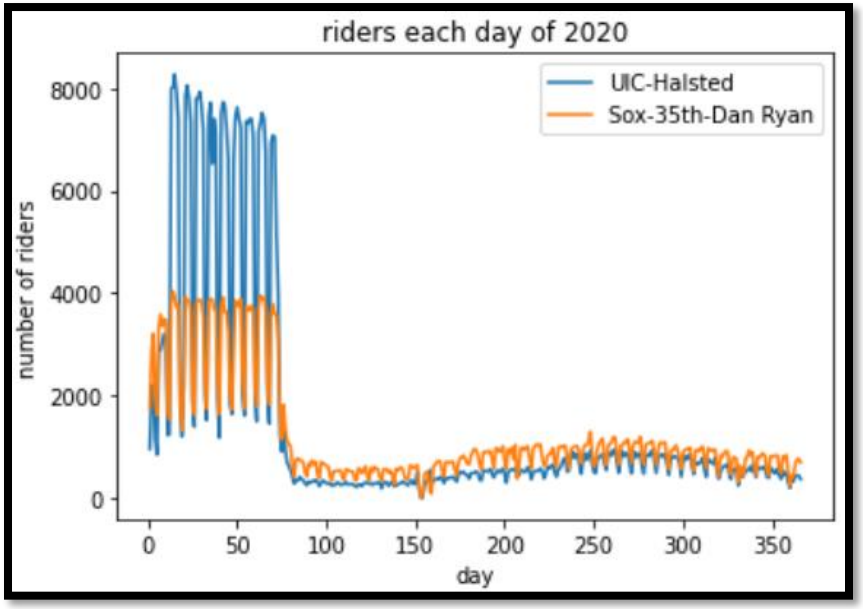
- **Programmatic access to a database**
- **Executing SQL from Python**
- **Examples: sqlite, MySQL**

Northwestern University

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**
- **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)
```

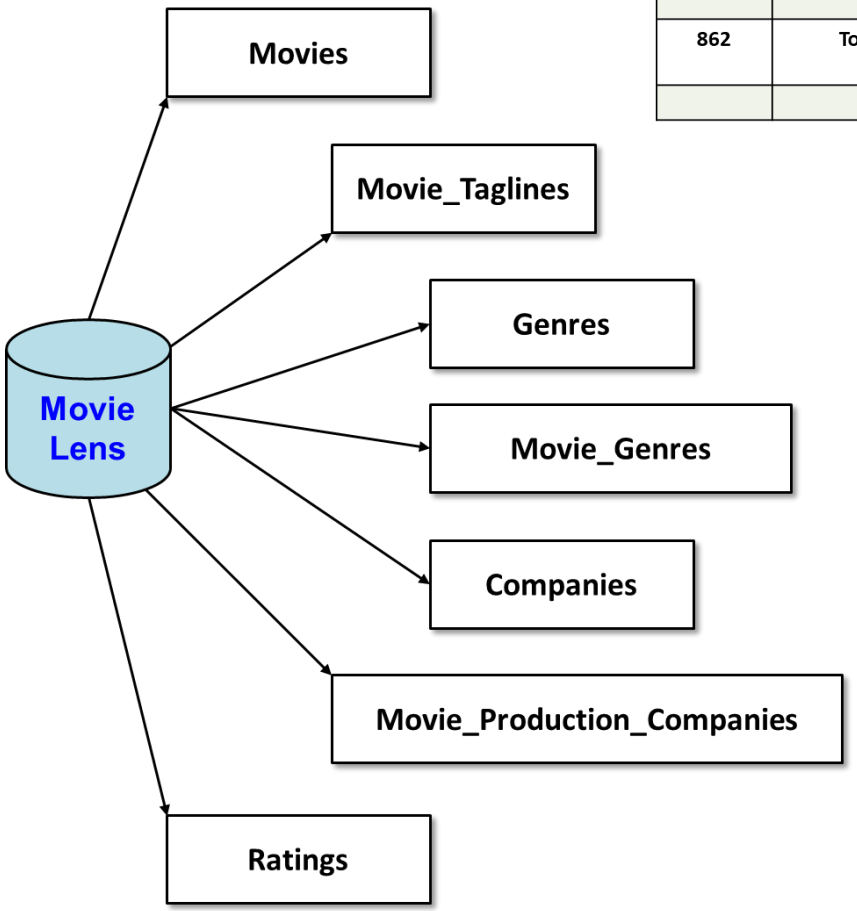
```
# query to retrieve N values from DB:
sql = """
      Select …
      From …
      Where … ;
      """

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

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

MovieLens database

Movie_ID	Title	Release_Date	Runtime	Original_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	373554033



Example

```
import sqlite3

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

sql = """
      select * from movies
      where title = 'Sleepless in Seattle';
      """

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

print(row)
```

```
docker> python3 test.py
(858, '1993-06-24 00:00:00.000', 105, 'en', 21000000, 227799884, 'Sleepless in Seattle')
docker>
```

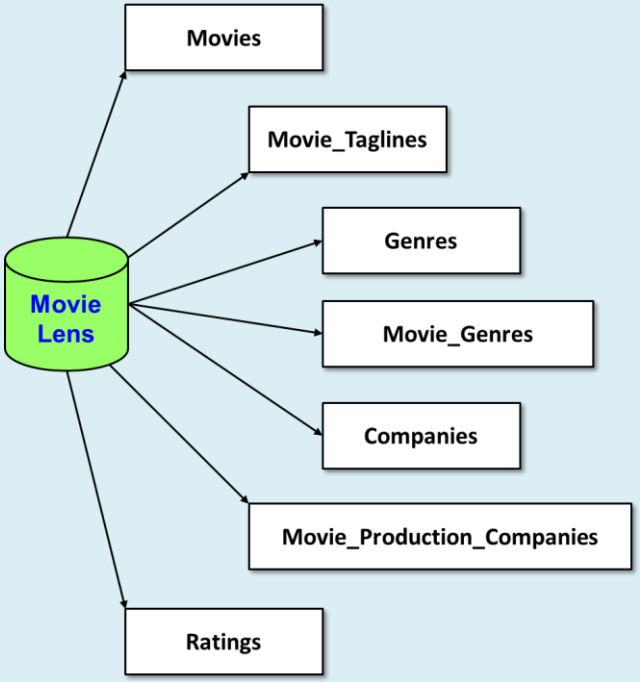
Top-10 movies in Drama genre

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]);
```

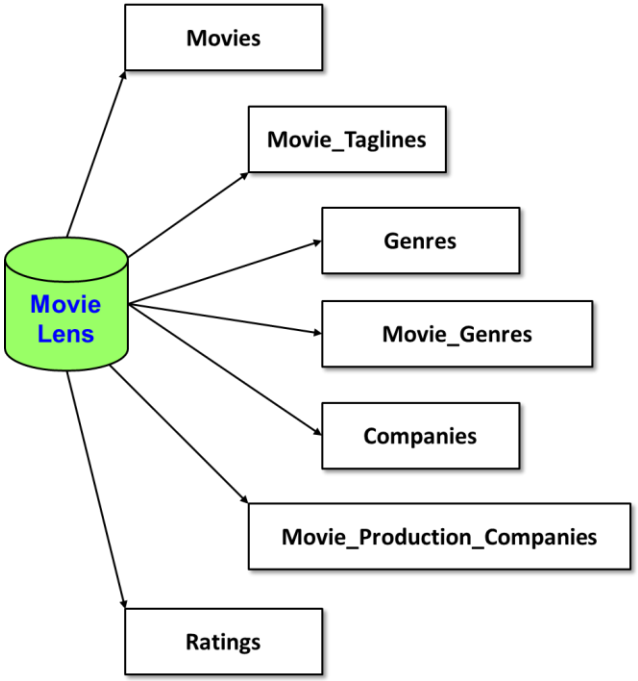



```
Movie: Sleepless in Seattle , avg rating: 8.98
Movie: The Million Dollar Hotel , avg rating: 8.97
Movie: Once Were Warriors , avg rating: 8.61
Movie: Confession of a Child of the Century , avg
rating: 8.47
Movie: The Thomas Crown Affair , avg rating: 8.47
Movie: Scarface , avg rating: 8.45
Movie: 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
```

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)
>
```

```python
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

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

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

How about executing SQL with MySQL?

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 ?

That's it, thank you!