Case_Study_SQL_Questions

November 22, 2024

1 Case Study on SQL

[267]:
```
import sqlite3
import pandas as pd
```

[268]:
```
def load_imdb_database(db_path):
  try:
    conn = sqlite3.connect(db_path)
    print("IMDB database loaded successfully.")
    return conn
  except sqlite3.Error as e:
    print(f"Error loading IMDB database: {e}")
    return None
```

[269]:
```
db_path = "imdb.db"
conn = load_imdb_database(db_path)

if conn:
  print("Connection successful")
  cursor = conn.cursor()
```

```
IMDB database loaded successfully.
Connection successful
```

1.1 Following is the schema of the IMDB database. It lists all tables and their schemas.

[270]:
```
cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

print("Database Schema:")
for table in tables:
    table_name = table[0]
    print(f"\nTable: {table_name}")

    cursor.execute(f"PRAGMA table_info({table_name});")
    schema = cursor.fetchall()
```

```
    print("Columns:")
    for column in schema:
        cid, name, dtype, notnull, dflt_value, pk = column
        print(f"  - {name} ({dtype}), NOT NULL: {bool(notnull)}, Default:␣
↪{dflt_value}, Primary Key: {bool(pk)}")
```

```
Database Schema:

Table: Movie
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - title (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - year (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - rating (REAL), NOT NULL: False, Default: None, Primary Key: False
  - num_votes (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: Genre
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - Name (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - GID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: Language
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - Name (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - LAID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: Country
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - Name (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - CID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: Location
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - Name (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - LID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: M_Location
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - LID (REAL), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False
```

```
Table: M_Country
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - CID (REAL), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: M_Language
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - LAID (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: M_Genre
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - GID (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: Person
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - PID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - Name (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - Gender (TEXT), NOT NULL: False, Default: None, Primary Key: False

Table: M_Producer
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - PID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: M_Director
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - PID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False

Table: M_Cast
Columns:
  - index (INTEGER), NOT NULL: False, Default: None, Primary Key: False
  - MID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - PID (TEXT), NOT NULL: False, Default: None, Primary Key: False
  - ID (INTEGER), NOT NULL: False, Default: None, Primary Key: False
```

2 Write SQL queries for the following questions

2.1 1. Write query to list first 5 rows of Person table

[271]:
```
q1 = "SELECT * FROM Person LIMIT 5;"
```

[272]:
```
def q1_grader(q1):
    result = pd.read_sql_query(q1, conn)
    return result.shape == (5, 4)
print(q1_grader(q1))
```

```
True
```

2.2 2. Write query to select title, year and rating from Movie table

[273]:
```
q2 = "SELECT title, year, rating FROM Movie;"
```

[274]:
```
def q2_grader(q2):
  result = pd.read_sql_query(q2, conn)
  return result.shape == (3473, 3) and result['title'][0]=='Mowgli'
print(q2_grader(q2))
```

```
True
```

2.3 3. Write query to get title of first movie in movie table sorted by year in ascending order

[275]:
```
q3 = "SELECT title FROM Movie ORDER BY year ASC LIMIT 1;"
```

[276]:
```
def q3_grader(q3):
  result = pd.read_sql_query(q3, conn)
  return result['title']=='Alam Ara'
print(q3_grader(q3))
```

```
0    True
Name: title, dtype: bool
```

2.4 4. Write query to get the very first year in which Devdas movie was released

[277]:
```
q4 = "SELECT year FROM Movie WHERE title='Devdas' ORDER BY year ASC LIMIT 1;"
```

[278]:
```
def q4_grader(q4):
  result = pd.read_sql_query(q4, conn)
  return result['year']=='1936'
print(q4_grader(q4))
```

```
0    True
Name: year, dtype: bool
```

2.5 5. Write query to get the number of movies released in 2018

[279]:
```
q5 = "SELECT COUNT(title) FROM Movie WHERE year=2018;"
```

[280]:
```
def q5_grader(q5):
  result = pd.read_sql_query(q5, conn)
  return result.iloc[0, 0] == 93
print(q5_grader(q5))
```

```
True
```

2.6 6. Write query to get the title of the movie with most number of votes in 2012

[281]:
```
q6 = "SELECT title FROM Movie WHERE year=2012 ORDER BY num_votes DESC LIMIT 1;"
```

[282]:
```
def q6_grader(q6):
  result = pd.read_sql_query(q6, conn)
  return result["title"]=="The Avengers"
print(q6_grader(q6))
```

```
0    True
Name: title, dtype: bool
```

2.7 7. Write SQL query to find all the unique movie titles released in 2018

[283]:
```
q7 = "SELECT DISTINCT title FROM Movie WHERE year=2018;"
```

[284]:
```
def q7_grader(q7):
  result = pd.read_sql_query(q7, conn)
  return result.shape==(93, 1)
print(q7_grader(q7))
```

```
True
```

2.8 8. Write SQL query to get total number of movies released between 2017 (inclusive) and 2018 (inclusive)

[285]:
```
q8 = "SELECT COUNT(title) FROM Movie WHERE year BETWEEN 2017 AND 2018;"
```

[286]:
```
def q8_grader(q8):
  result = pd.read_sql_query(q8, conn)
  return result.iloc[0, 0] == 211
print(q8_grader(q8))
```

```
True
```

2.9 9. Write SQL query to find the year in which maximum number of movies released

[287]:
```
q9 = "SELECT year FROM Movie GROUP BY year ORDER BY COUNT(year) DESC LIMIT 1 ;"
```

[288]:
```
def q9_grader(q9):
    result = pd.read_sql_query(q9, conn)
    return result["year"][0]=="2005"
print(q9_grader(q9))
```

```
True
```

2.10 10. Write SQL query to find the title of the movie with rating>9.5 and number of votes > 90

[289]:
```
q10 = "SELECT title FROM Movie WHERE rating>9.5 AND num_votes>90;"
```

[290]:
```
def q10_grader(q10):
  result = pd.read_sql_query(q10, conn)
  return result["title"][0]=="Man on Mission Fauladi"
print(q10_grader(q10))
```

```
True
```

2.11 11. Write SQL query to find the number of movies which has the word 'Dilwale' in their title

[291]:
```
q11 = "SELECT COUNT(title) FROM Movie WHERE title LIKE '%Dilwale%';"
```

[292]:
```
def q11_grader(q11):
  result = pd.read_sql_query(q11, conn)
  return result.iloc[0, 0] == 4
print(q11_grader(q11))
```

```
True
```

2.12 12. Write nested SQL query to find the CID of country which produced most number of movies

[293]:
```
q12 = "SELECT DISTINCT CID FROM M_Country WHERE CID IN (SELECT CID FROM␣
 ↪M_Country GROUP BY CID ORDER BY COUNT(CID) DESC LIMIT 1);"
```

[294]:
```
def q12_grader(q12):
    result = pd.read_sql_query(q12, conn)
    return result.iloc[0]==2.0
print(q12_grader(q12))
```

```
CID    True
Name: 0, dtype: bool
```

2.13 13. Write nested SQL query to the country which produced most number of movies (use both Courty table and M_Country table)

[295]:
```
q13 = "SELECT Name FROM Country WHERE CID IN (SELECT CID FROM M_Country GROUP␣
 ↪BY CID ORDER BY COUNT(CID) DESC LIMIT 1);"
```

[296]:
```
def q13_grader(q13):
  result = pd.read_sql_query(q13, conn)
  return result.iloc[0, 0] == "India"
print(q13_grader(q13))
```

```
True
```

2.14 14. Write SQL query to get the year and number of movies per year having number of movies per year is greater than 100

[297]:
```
q14 = "SELECT DISTINCT year, COUNT(title) as num_of_movies FROM Movie GROUP BY␣
 ↪year HAVING num_of_movies>100;"
```

[298]:
```
def q14_grader(q14):
  result = pd.read_sql_query(q14, conn)
  return result.shape==(13,2)
print(q14_grader(q14))
```

```
True
```

2.15 15. Write SQL query to get the Name and Language ID (LAID) corresponding to Malayalam language

[299]:
```
q15 = "SELECT Name, LAID FROM Language WHERE Name='Malayalam';"
```

[300]:
```
def q15_grader(q15):
  result = pd.read_sql_query(q15, conn)
  return result[["Name", "LAID"]].values.tolist() == [['Malayalam', 19]]
print(q15_grader(q15))
```

```
True
```

2.16 16. Write SQL query to do inner join with movie table and M_Language table with MID colums

[301]:
```
q16 = "SELECT * FROM Movie AS M INNER JOIN M_Language AS ML ON M.MID=ML.MID;"
```

[302]:
```
def q16_grader(q16):
  result = pd.read_sql_query(q16, conn)
  return result
print(q16_grader(q16))
```

```
      index        MID          title  year  rating  num_votes  index  \
0         0  tt2388771         Mowgli  2018     6.6      21967      0
1         1  tt5164214  Ocean's Eight  2018     6.2     110861      1
2         2  tt1365519    Tomb Raider  2018     6.4     142585      2
3         3  tt0848228   The Avengers  2012     8.1    1137529      3
4         4  tt8239946        Tumbbad  2018     8.5       7483      4
…       …        …              …     …    …      …         …     …
3468   3470  tt0090611    Allah-Rakha  1986     6.2         96   3470
3469   3471  tt0106270          Anari  1993     4.7        301   3471
3470   3472  tt0852989  Come December  2006     5.7         57   3472
3471   3473  tt0375882     Kala Jigar  1939     3.3        174   3473
3472   3474  tt0375890         Kanoon  1994     3.2        103   3474

            MID  LAID    ID
0     tt2388771     0     0
1     tt5164214     0     1
2     tt1365519     0     2
3     tt0848228     0     3
4     tt8239946     1     4
…         …   …   …
3468  tt0090611     2  3470
3469  tt0106270     2  3471
3470  tt0852989     2  3472
3471  tt0375882     2  3473
3472  tt0375890     2  3474

[3473 rows x 10 columns]
```

2.17 17. Write SQL query to list title, year and rating of malayalam movies in the database by doing an inner join with movie table and M_Language table with MID column, also assuming language ID of malayalam movies as 19

[303]:
```
q17 = "SELECT title, year, rating FROM Movie AS M INNER JOIN M_Language AS ML␣
 ↪ON M.MID=ML.MID WHERE LAID=19;"
```

[304]:
```
def q17_grader(q17):
  result = pd.read_sql_query(q17, conn)
  return result.shape==(16,3)
print(q17_grader(q17))
```

```
True
```

[305]:
```
![ -f academic.db ] && rm academic.db

conn = sqlite3.connect("academic.db")
cursor = conn.cursor()
```

2.18 18. Write SQL query to Create a table named students with two columns id (integer type) and name (varchar type)

[306]:
```
q18a = "CREATE TABLE Students ( id int, name varchar);"
```

[307]:
```
def q18_grader_a(q18):
  try:
    cursor.execute(q18)
    return True
  except:
    pass

def q18_grader_b(q19):
  result = pd.read_sql_query(q19, conn)
  return result.columns.tolist() == ['id', 'name']

q18b = """SELECT * FROM students;"""

print(q18_grader_a(q18a) and q18_grader_b(q18b))
```

```
True
```

2.19 19. Write SQL query to insert the values (1, 'Alice') into students table

[308]:
```
q19_a = "INSERT INTO Students VALUES (1,'Alice');"
```

[309]:
```
def q19_grader_a(q19_a):
  try:
    cursor.execute(q19_a)
  except:
    pass
  return True

def q19_grader_b(q19_b):
  result = pd.read_sql_query(q19_b, conn)
  return result.values.tolist() == [[1, 'Alice']]

q19_b = "SELECT * FROM students"
print(q19_grader_a(q19_a) and q19_grader_b(q19_b))
```

```
True
```

2.20 20. Write SQL Query to add the following more information to students table

ID	Name
2	Bob
3	Charlie

[310]:
```
q20 = "INSERT INTO Students VALUEs (2,'Bob'),(3,'Charlie');"
```

[311]:
```
def q20_grader(q20):
  try:
    cursor.execute(q20)
  except:
    pass
  return True

def q20_grader_b(q20_b):
  result = pd.read_sql_query(q20_b, conn)
  return result.values.tolist() == [[1, 'Alice'], [2, 'Bob'], [3, 'Charlie']]

print(q20_grader(q20) and q20_grader_b(q19_b))
```

```
True
```

[312]:
```
conn.close()
```