**02/14/2025**

select \* from movies2.data\_movies;

-- Single line comment

/\*

Which genre has the highest average runtime?

\*/

SELECT genre, AVG(runtime) AS AverageRunTime

FROM movies2.data\_movies

GROUP BY genre

ORDER BY AverageRunTime DESC

LIMIT 5; -- LIMIT 1; for highest

-- ==============================================

/\*

Which country produced the most movies in 1988

\*/

SELECT country, COUNT(country) AS movie\_count

FROM movies2.data\_movies

WHERE year = 1988 -- Change to view for a different year

GROUP BY country

ORDER BY movie\_count DESC

LIMIT 1;

-- ==============================================

/\*

Which director has the most films produced in unique years?

\*/

SELECT COUNT( DISTINCT year) as DistinctYear, director

FROM movies2.data\_movies

GROUP BY director

ORDER BY DistinctYear DESC

LIMIT 5;

-- ==============================================

/\*

What are the most common runtimes (rounded to the nearest multiple of ten)?

\*/

SELECT (FLOOR(runtime / 10) \* 10) AS RoundedRuntime, COUNT((FLOOR(runtime / 10) \* 10)) AS NumberOfMovies

FROM movies2.data\_movies

GROUP BY (FLOOR(runtime / 10) \* 10)

ORDER BY NumberOfMovies DESC

LIMIT 1;

-- ==============================================

/\*

What low-scored movie had the highest gross collection?

\*/

SELECT name, floor(score) as RoundScore, gross

FROM movies2.data\_movies

-- WHERE score = 2

ORDER BY RoundScore ASC, gross DESC

-- LIMIT 5

;

-- Single line comment

/\*

Which genre has the highest average runtime

\*/

SELECT genre, AVG(runtime) AS AverageRT

FROM movies2.data\_movies

GROUP BY genre

ORDER BY AverageRT DESC

LIMIT 5;

SELECT genre, AVG(runtime) -- AS AverageRT

FROM movies2.data\_movies

GROUP BY genre

ORDER BY AVG(runtime) DESC

LIMIT 5;

/\*

Which low score movie had the highest gross collection -> we floor of the score

\*/

select name, floor(score) AS RoundScore, gross

from movies2.data\_movies

where RoundScore = 2;

-- order by RoundScore ASC, gross DESC

-- limit 1;

-- group by RoundScore;

/\*

number of movies per rating that grossed double the budget despite a score lower than average

\*/

select \* from movies2.data\_movies;

— --------------------------------------------------------------------------------------------------------------------------

**02/14/2025**

import mysql.connector

import pandas as pd

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

conn = mysql.connector.connect(

host = "localhost",

user = "root",

password = "root123",

database = "thisfellow"

)

if conn.is\_connected():

print("Established")

query\_1 = "Select \* FROM student WHERE Major = 'CS';"

data\_1 = pd.read\_sql(query\_1, conn)

print(data\_1.head())

plt.figure(figsize = (10,6))

plt.bar(data\_1['Name'], data\_1['GPA'])

plt.xlabel('Student Name')

plt.ylabel('GPA')

plt.xticks(rotation = 45)

plt.title("GPA of CS major students")

plt.tight\_layout()

plt.show()

—------------------------------------------------------------------------------------------------------------------

**02/21/2025**

import mysql.connector

import pandas as pd

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

conn = mysql.connector.connect(

host="localhost", # e.g., "localhost"

user="root", # e.g., "root"

password="root123", # your MySQL password

database="sql\_activity1" # the name of your database

)

if conn.is\_connected():

print("established")

# Query 2.3

query\_1 = "SELECT ProductID, SUM(Quantity) AS TotalOrdered FROM Orders GROUP BY ProductID;"

# Find products that have never been ordered

query\_2 = """

SELECT Products.ProductName

FROM Orders

RIGHT JOIN Products ON Orders.ProductID = Products.ProductID

WHERE Orders.OrderID IS NULL;

"""

# Use pandas to execute the query and store the results in a DataFrame

data\_1 = pd.read\_sql(query\_1, conn)

data\_2 = pd.read\_sql(query\_2, conn)

print(data\_1.head())

print(data\_2)

# Close the connection

#conn.close()

**02/25/2025**

import mysql.connector

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.cm as cm

import warnings

warnings.filterwarnings('ignore')

def plot\_bar\_graph(data, x\_axis, y\_axis, x\_label, y\_label, title):

# Function to plot a bar graph

# Parameters:

# data (dtype -> DataFrame)

# column on x-axis (dtype -> string)

# colum on y-axis (dtype -> string)

# x-axis label (dtype -> string)

# y-axis label (dtype -> string)

# title for the plot (dtype -> string)

plt.figure(figsize=(10, 6))

colors = cm.viridis(np.linspace(0, 1, len(data[x\_axis])))# Generate colors based on the number of bars

plt.bar(data[x\_axis], data[y\_axis], color = colors)

plt.xlabel(x\_label)

plt.ylabel(y\_label)

plt.title(title)

plt.xticks(rotation=45) # Rotate x-axis labels for better readability if needed

plt.tight\_layout() # Adjust layout to prevent clipping of labels

plt.show()

conn = mysql.connector.connect(

host="localhost", # e.g., "localhost"

user="root", # e.g., "root"

password="root123", # your MySQL password

database="sql\_activity1" # the name of your database

)

if conn.is\_connected():

print("established")

# Query 2.3

query\_1 ="""

SELECT ProductID, SUM(Quantity) AS TotalOrdered

FROM Orders

GROUP BY ProductID;

"""

# Use pandas to execute the query and store the results in a DataFrame

data\_1 = pd.read\_sql(query\_1, conn)

#print(data\_1.head())

#plot\_bar\_graph(data\_1, 'ProductID', 'TotalOrdered', 'Products', 'Quantity', 'Total Quantities of Products sold')

# Retrieve the product names along with their total quantities ordered

query\_2 = """

SELECT ProductName, SUM(Quantity) AS TotalOrdered

FROM products

JOIN orders ON products.ProductID = orders.ProductID

GROUP BY ProductName;

"""

pd.set\_option('display.max\_rows', None)

pd.set\_option('display.max\_columns', None)

pd.set\_option('display.max\_colwidth', None)

data\_2 = pd.read\_sql(query\_2, conn)

print(data\_2.head())

plot\_bar\_graph(data\_2, 'ProductName', 'TotalOrdered', 'Products', 'Quantity', 'Total Quantities of Products sold')

# Count of Orders by Order Status

# This query counts the number of orders under each OrderStatus category.

# Total Quantity of Products Sold Per Category

# This query calculates the total number of products sold in each category.

**03/04/2025**

SELECT \* FROM one\_to\_one.passport;

SELECT \* FROM one\_to\_one.person;

-- Delete cascade

start transaction;

-- If I delete a row in person table

DELETE FROM person

WHERE person\_id = 2;

-- SELECT \* FROM one\_to\_one.person;

-- It should also reflect in passport table

SELECT \* FROM one\_to\_one.passport;

rollback;

-- Update cascade

start transaction;

-- If I update a rwo in person table,

UPDATE person

SET person\_id = 100

WHERE person\_id = 2;

-- It should be reflected in passport table

SELECT \* FROM one\_to\_one.passport;

rollback;

/\*

In MySQL, auto-increment counters do not revert to their previous values upon ROLLBACK.

Once an auto-increment value is used (even if the corresponding row gets rolled back and never committed),

the “counter” moves forward and stays there. Therefore, you’ll see a gap in the sequence if you start a transaction,

insert a row, roll it back, and then insert another row later.

\*/