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.
 */
```