PM SHRI KENDRIYA VIDYALAYA OEF HAZRATPUR

PRACTICAL WORK BOOK

INFORMATICS PRACTICES

Class XII | Subject Code: 065 | Session 2026-27

Covers: Pandas Series & DataFrame | Data Visualization with Matplotlib
Python – MySQL Connectivity (Import / Export) | SQL Queries & Functions
Board-Pattern Programming Questions with Solutions

Name of Student	
Class & Section	
Roll Number	
School Name	PM SHRI KENDRIYA VIDYALAYA OEF HAZRATPUR
Session	2026 – 2027
Subject Teacher	MR. DEEPAK KUMAR SHARMA

CERTIFICATE

This is to certify that _____ of Class XII, Section _____, Roll No. _____ has successfully completed the Practical Work Book for the subject Informatics Practices (Code 065) as prescribed by the Central Board of Secondary Education (CBSE) for the academic session 2026-27, during the year _____.

The practical work comprises programs based on Pandas Series and DataFrame, Data Visualization using Matplotlib, Python-MySQL connectivity (import/export of data), and MySQL queries using aggregate functions, grouping and joins, carried out under my supervision in the school computer laboratory.

Date: _____

Place: _____

_____
Signature of Subject Teacher
Internal Examiner

_____
Signature of External Examiner

Syllabus Mapping (CBSE IP 065, Class XII, 2026-27)

Unit / Skill (as per CBSE Curriculum)	Covered in this Work Book
Pandas — Series: creation, indexing, slicing, head/tail, arithmetic operations	Part A: Programs 1–6
Pandas — DataFrame: creation, selection (loc/iloc), add/delete columns & rows, filtering, sorting, missing data, groupby, merge, concat, apply(), rename	Part A: Programs 7–21
Data Visualization using Matplotlib: line, bar, histogram, pie, scatter, box plot, customization	Part A: Programs 22–28
Integrated Pandas + Matplotlib mini-projects	Part A: Programs 29–30
Interface of Python with an SQL Database — connecting, creating tables, CRUD operations, executemany()	Part B: Programs 31–37
Import/Export of data between Pandas DataFrame and MySQL	Part B: Programs 38–40
SQL revision: DDL, DML, constraints, DISTINCT, ORDER BY	Part C: Queries 41–50
SQL Aggregate functions: COUNT, SUM, AVG, MAX, MIN; GROUP BY, HAVING	Part C: Queries 51–55
Mathematical, String and Date functions in MySQL	Part C: Queries 56–58
Querying two tables using Equi-Join	Part C: Queries 59–60
Board-pattern programming/SQL questions (output prediction, error correction, case-study)	Part D: Q1–Q10

Note: Programs are arranged strand-wise and in increasing order of difficulty within each part, so that each program builds on the concepts practiced in the ones before it.

Index / Table of Contents

Part A — Python: Series, DataFrame & Data Visualization

27	Plot a Box Plot		
28	Customize a Chart (Title, Labels, Legend, Grid, Save)		
29	Mini Project: Analyse a Student Marks DataFrame with Statistics and Chart		
30	Mini Project: Analyse Sales Data from CSV using groupby() and Line Chart		

Part B — Python–MySQL Connectivity (Import/Export)

S.No	Title of Program / Query / Question	Page No.	Teacher's Sign
31	Connect Python to MySQL Database		
32	Create a Table in MySQL from Python		
33	Insert a Single Record into MySQL from Python		
34	Insert Multiple Records using executemany()		
35	Fetch and Display Records from MySQL in Python		
36	Update Records in MySQL Table from Python		
37	Delete Records from MySQL Table from Python		
38	Export Data from a Pandas DataFrame to a MySQL Table		
39	Import Data from a MySQL Table into a Pandas DataFrame		
40	Mini Project: Import a CSV File into a MySQL Table using Python		

Part C — MySQL Queries and Functions

S.No	Title of Program / Query / Question	Page No.	Teacher's Sign
41	Create Database and Table with Constraints		
42	Insert Records into the Table		
43	Display All Records - SELECT *		
44	SELECT with WHERE Clause		
45	UPDATE Command		
46	DELETE Command		
47	ALTER TABLE - Add a Column		
48	ALTER TABLE - Modify a Column		
49	Use of DISTINCT		
50	ORDER BY (Ascending / Descending)		
51	Aggregate Function - COUNT()		
52	Aggregate Functions - SUM() and AVG()		
53	Aggregate Functions - MAX() and MIN()		
54	GROUP BY Clause		
55	GROUP BY with HAVING Clause		
56	String Functions - UPPER, LOWER, LENGTH, SUBSTRING/MID		
57	Date Functions - NOW(), DATE(), MONTH(), YEAR()		
58	Math Functions - ROUND(), POWER(), MOD()		
59	Equi-Join of Two Tables		
60	Complex Query: JOIN + GROUP BY + HAVING + ORDER BY		

Part D — Board-Pattern Programming Questions

S.No	Title of Program / Query / Question	Page No.	Teacher's Sign
1	[Predict the Output (Series)]		
2	[Find and Correct the Errors (DataFrame)]		
3	[DataFrame Output-Based Question]		
4	[Write SQL Queries on a Given Table]		
5	[MySQL - Python Connectivity Output-Based Question]		
6	[Predict the Output (Aggregate Functions in SQL)]		
7	[Case-Study Based Question (DataFrame + Visualization)]		
8	[Assertion-Reason / Conceptual Question]		
9	[Write a Python-MySQL Program (Full Program Question)]		
10	[Equi-Join Based Board Question]		

PART A — Python Programs: Pandas Series, DataFrame & Data Visualization (Matplotlib)

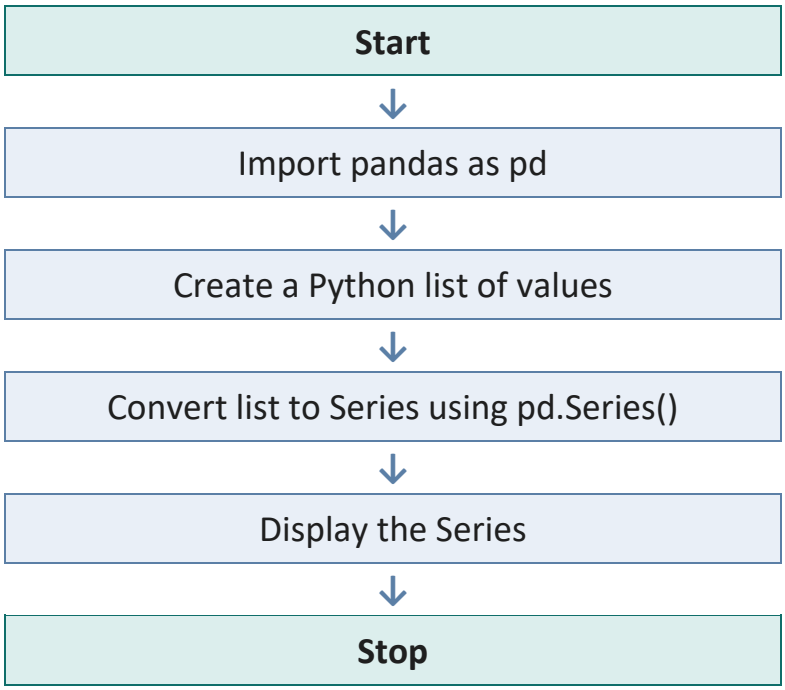
30 Programs, arranged in increasing order of difficulty: Series (1–6) → DataFrame (7–21) → Matplotlib Visualization (22–28) → Integrated Mini-Projects (29–30)

Program 1: Create a Series from a Python List

Aim:

To create a Pandas Series from a Python list and display it.

Flow Chart:



Program (Source Code):

```
import pandas as pd

data = [10, 20, 30, 40, 50]
s = pd.Series(data)
print("Series from list:")
print(s)
```

Output:

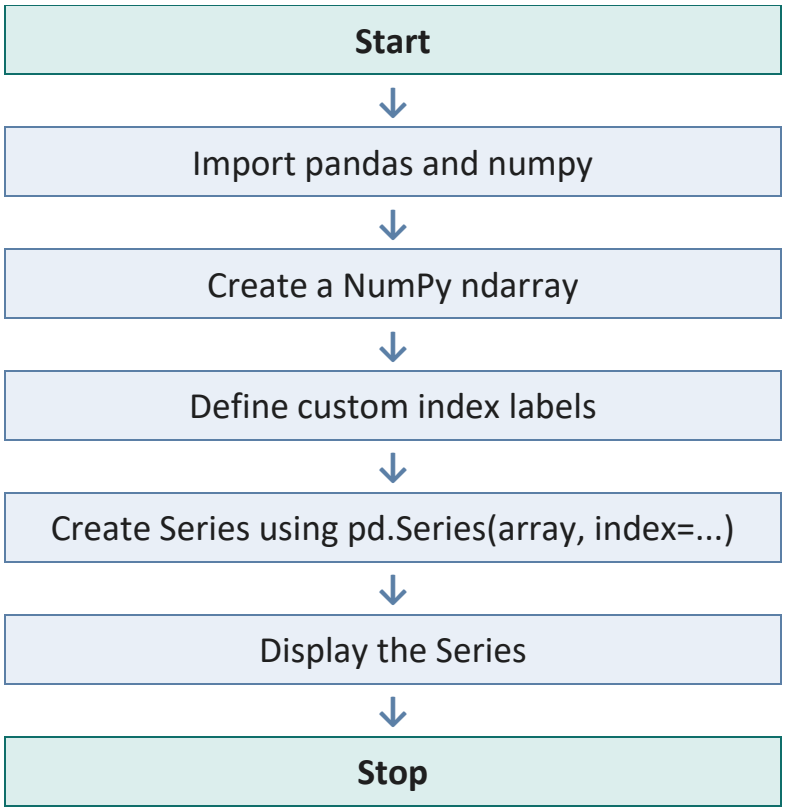
```
Series from list:
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

Program 2: Create a Series from a NumPy ndarray

Aim:

To create a Pandas Series from a NumPy array with a custom index.

Flow Chart:



Program (Source Code):

```
import pandas as pd
import numpy as np

arr = np.array([15, 25, 35, 45])
s = pd.Series(arr, index=['a', 'b', 'c', 'd'])
print(s)
```

Output:

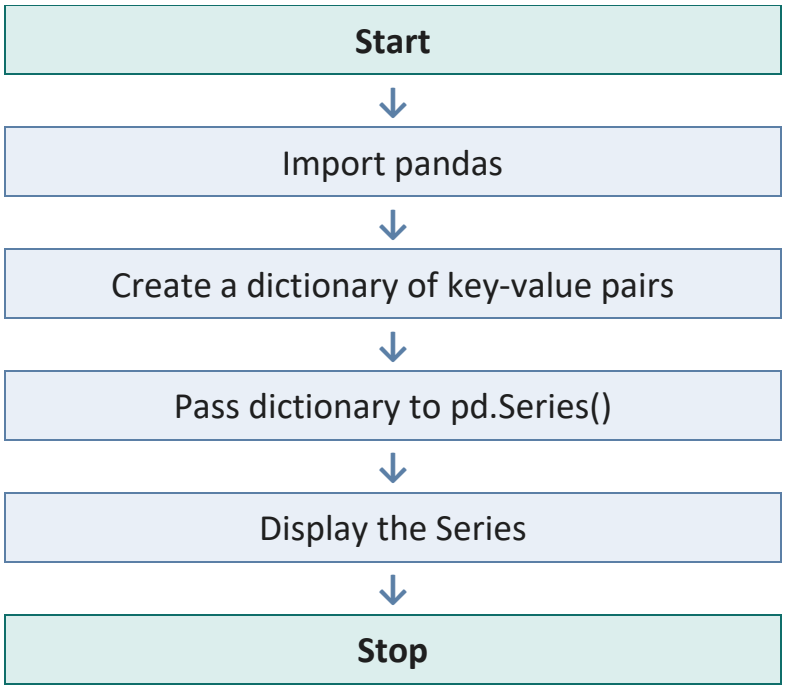
```
a    15
b    25
c    35
d    45
dtype: int64
```

Program 3: Create a Series from a Dictionary

Aim:

To create a Pandas Series from a Python dictionary where keys become the index.

Flow Chart:



Program (Source Code):

```
import pandas as pd

marks = {'Physics': 88, 'Chemistry': 92, 'Maths': 95}
s = pd.Series(marks)
print(s)
```

Output:

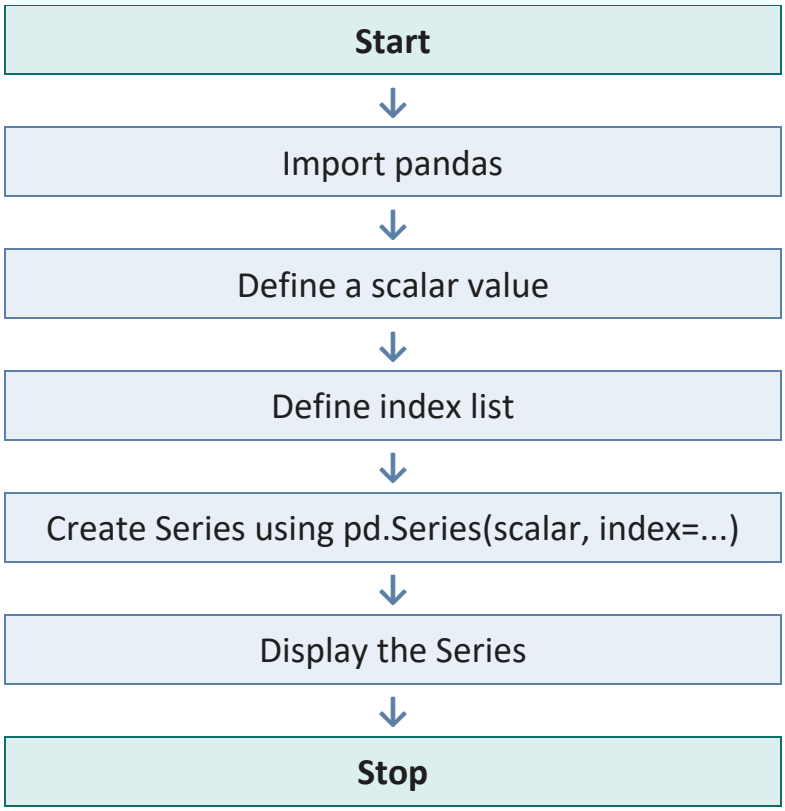
```
Physics      88
Chemistry    92
Maths        95
dtype: int64
```

Program 4: Create a Series from a Scalar Value

Aim:

To create a Pandas Series by repeating a single scalar value across a given index.

Flow Chart:



Program (Source Code):

```
import pandas as pd

s = pd.Series(7, index=[0, 1, 2, 3])
print(s)
```

Output:

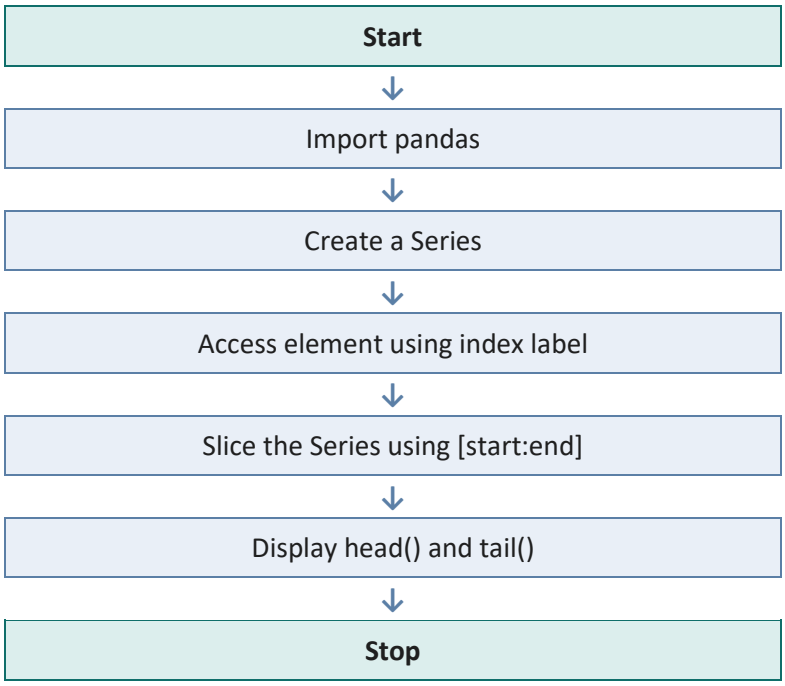
```
0    7
1    7
2    7
3    7
dtype: int64
```

Program 5: Series Indexing, Slicing, head() and tail()

Aim:

To access elements of a Series using indexing and slicing, and to display first/last records.

Flow Chart:



Program (Source Code):

```
import pandas as pd

s = pd.Series([12, 24, 36, 48, 60, 72], index=list('abcdef'))
print("Element at 'c':", s['c'])
print("Slice [1:4]:\n", s[1:4])
print("head(2):\n", s.head(2))
print("tail(2):\n", s.tail(2))
```

Output:

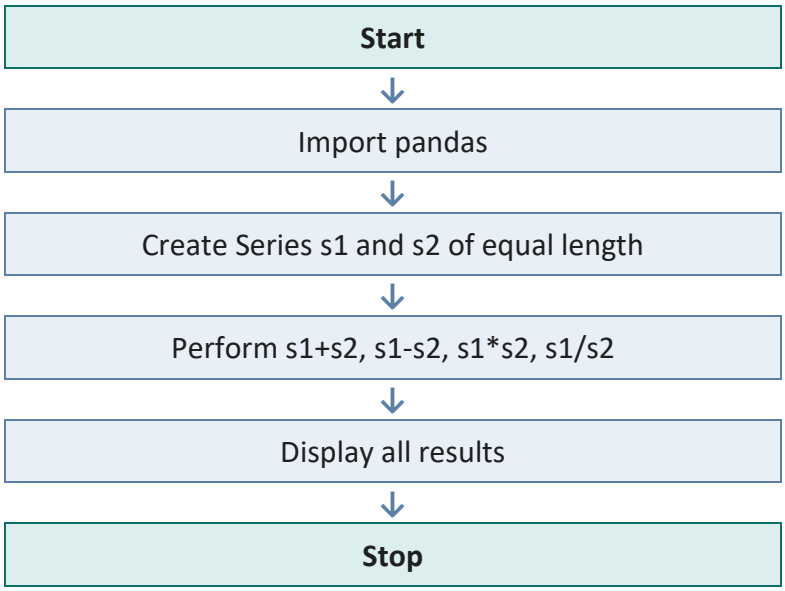
```
Element at 'c': 36
Slice [1:4]:
 b    24
c    36
d    48
dtype: int64
head(2):
 a    12
b    24
dtype: int64
tail(2):
 e    60
f    72
dtype: int64
```

Program 6: Arithmetic Operations on Two Series

Aim:

To perform addition, subtraction, multiplication and division on two Pandas Series.

Flow Chart:



Program (Source Code):

```
import pandas as pd

s1 = pd.Series([10, 20, 30])
s2 = pd.Series([1, 2, 3])
print("Addition:\n", s1 + s2)
print("Subtraction:\n", s1 - s2)
print("Multiplication:\n", s1 * s2)
print("Division:\n", s1 / s2)
```

Output:

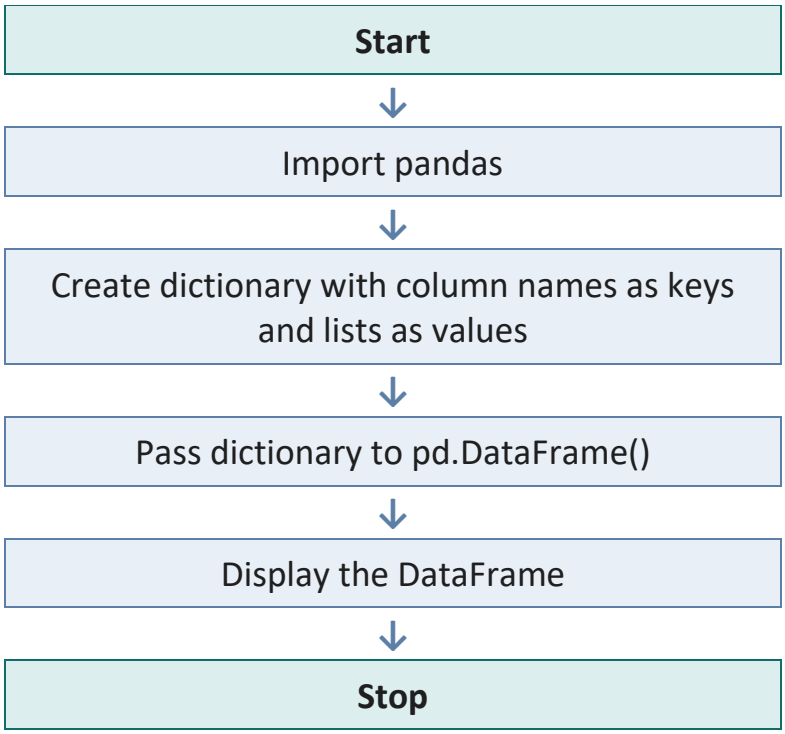
```
Addition:
 0    11
1    22
2    33
dtype: int64
Subtraction:
 0    9
1    18
2    27
dtype: int64
Multiplication:
 0    10
1    40
2    90
dtype: int64
Division:
 0    10.0
1    10.0
2    10.0
dtype: float64
```

Program 7: Create a DataFrame from a Dictionary of Lists

Aim:

To create a Pandas DataFrame using a dictionary whose values are lists.

Flow Chart:



Program (Source Code):

```
import pandas as pd

data = {'Name': ['Aman', 'Riya', 'Karan'],
        'Marks': [88, 92, 79],
        'City': ['Delhi', 'Amritsar', 'Pune']}
df = pd.DataFrame(data)
print(df)
```

Output:

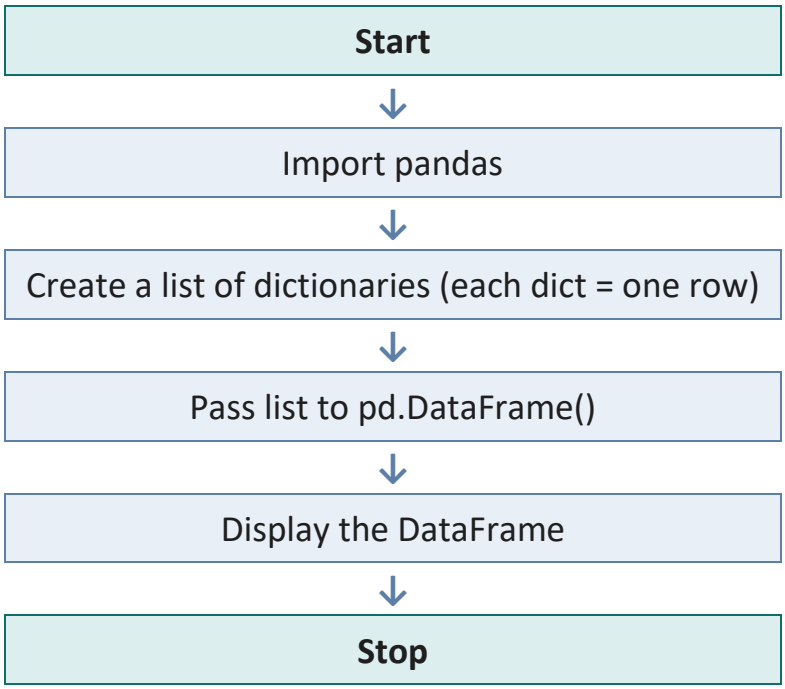
```
    Name  Marks      City
0   Aman     88     Delhi
1   Riya     92  Amritsar
2  Karan     79      Pune
```

Program 8: Create a DataFrame from a List of Dictionaries

Aim:

To create a Pandas DataFrame from a list where each element is a dictionary representing one row.

Flow Chart:



Program (Source Code):

```
import pandas as pd

data = [{'Roll': 1, 'Name': 'Simran', 'Grade': 'A'},
        {'Roll': 2, 'Name': 'Ishaan', 'Grade': 'B'}]
df = pd.DataFrame(data)
print(df)
```

Output:

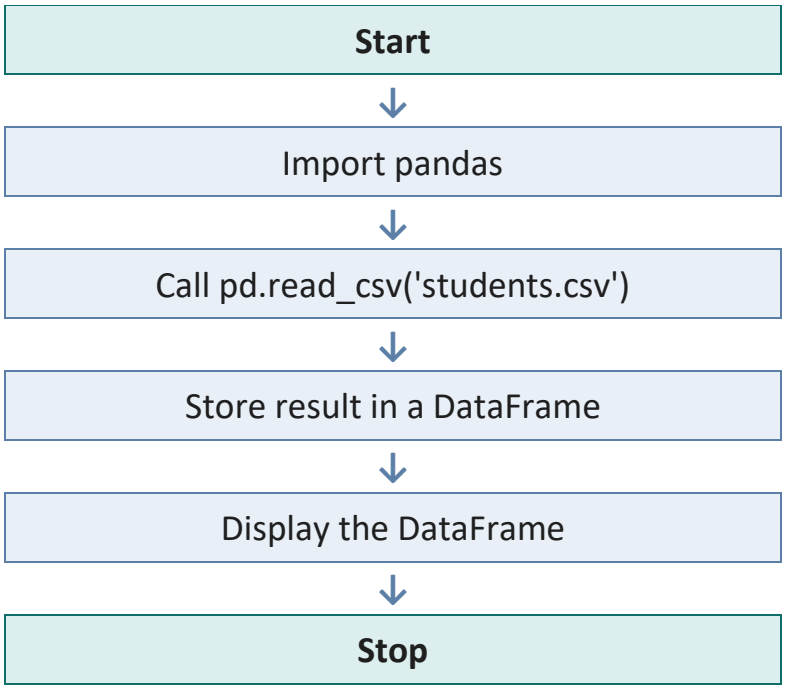
```
   Roll    Name Grade
0     1  Simran     A
1     2  Ishaan     B
```

Program 9: Create a DataFrame from a CSV File

Aim:

To read data from a CSV file into a Pandas DataFrame using read_csv().

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.read_csv('students.csv')
print(df)
```

Output:

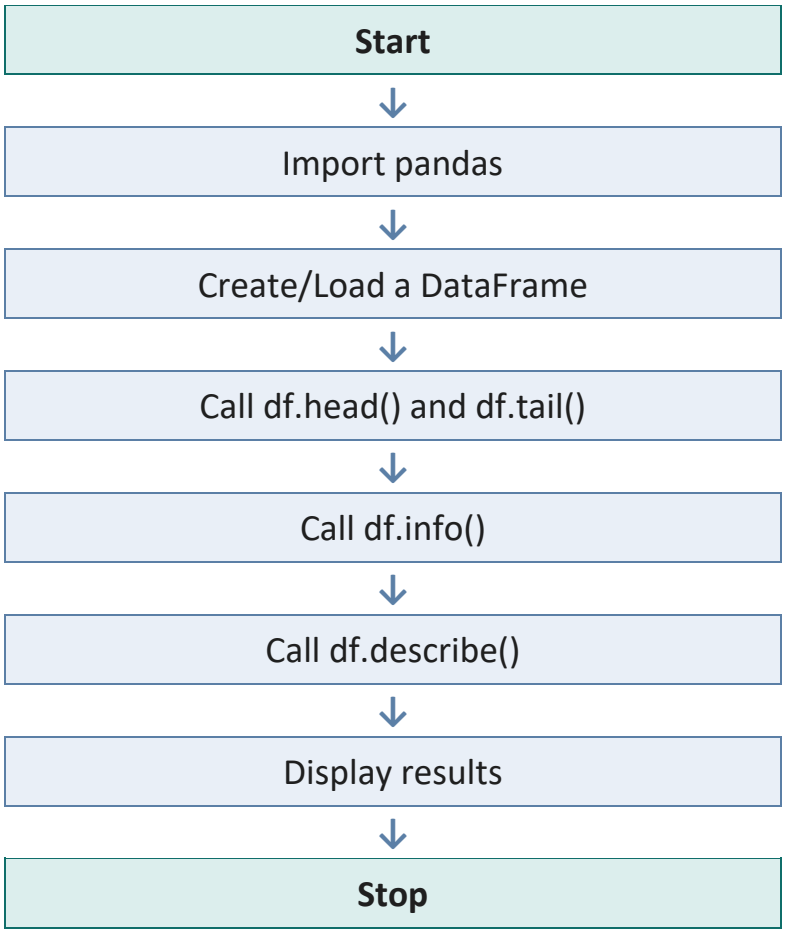
```
(Contents of students.csv are displayed as a DataFrame,
 e.g.:)
   RollNo    Name  Marks
0       1   Aditi     91
1       2   Rohan     85
2       3   Meher     78
```

Program 10: Display head(), tail(), info() and describe() of a DataFrame

Aim:

To explore a DataFrame's structure and summary statistics using built-in functions.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Item': ['Pen', 'Pencil', 'Eraser',
'Ruler'],
                   'Price': [10, 5, 3, 8],
                   'Qty': [100, 200, 150, 90]})
print(df.head(2))
print(df.tail(2))
print(df.info())
print(df.describe())
```

Output:

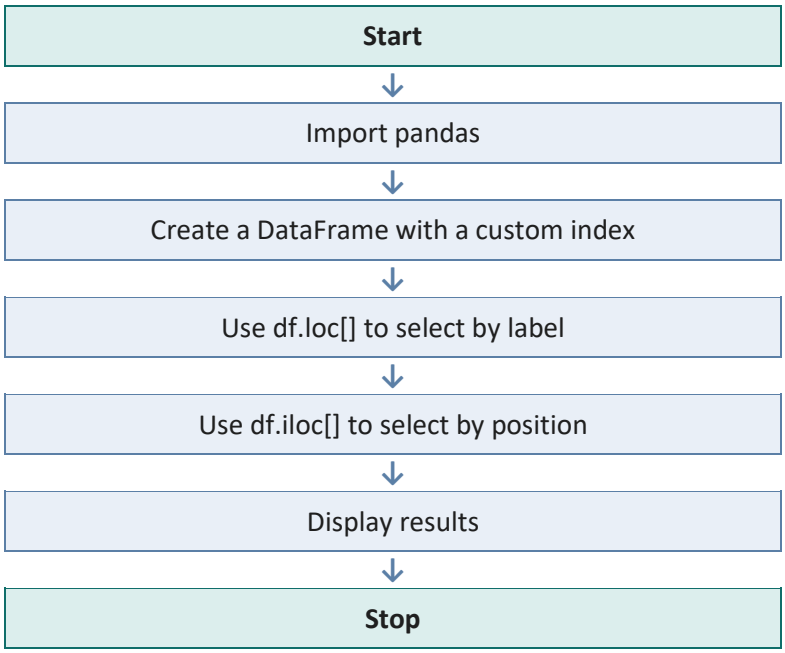
```
(head/tail show selected rows; info() shows column
 dtypes & non-null counts; describe() shows count,
 mean, std, min, 25%, 50%, 75%, max for numeric columns)
```

Program 11: Select Rows/Columns using loc[] and iloc[]

Aim:

To access specific rows and columns of a DataFrame using label based (loc) and position based (iloc) indexing.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Name': ['A', 'B', 'C', 'D'],
                   'Marks': [67, 78, 89, 95]},
index=['r1','r2','r3','r4'])
print(df.loc['r2'])
print(df.loc['r1':'r3', 'Marks'])
print(df.iloc[1])
print(df.iloc[0:2, 0:2])
```

Output:

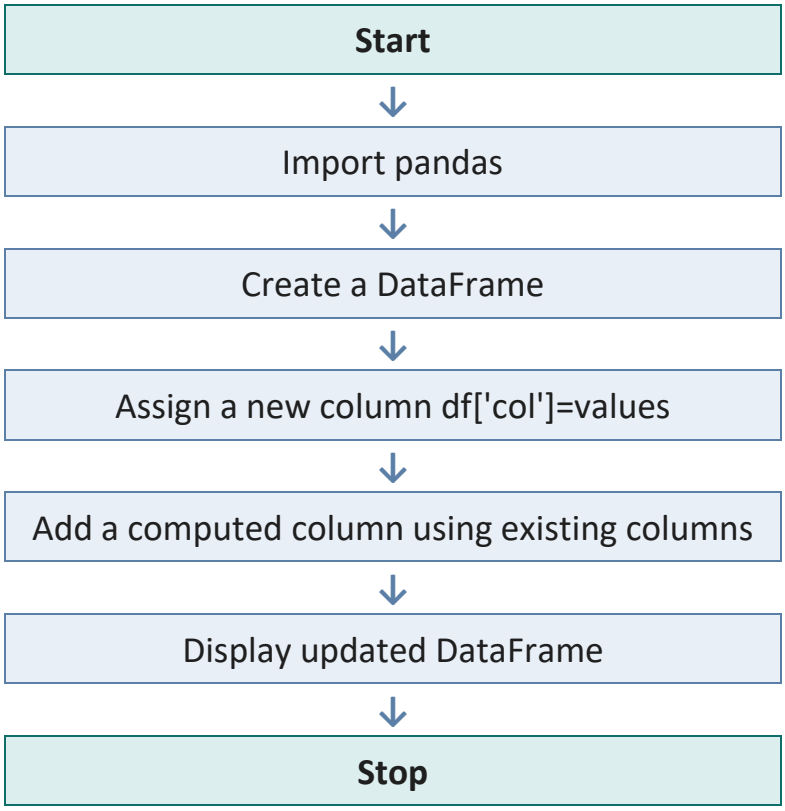
```
Name        B
Marks      78
Name: r2, dtype: object
r1    67
r2    78
r3    89
Name: Marks, dtype: int64
Name        B
Marks      78
Name: r2, dtype: object
  Name  Marks
r1    A     67
r2    B     78
```

Program 12: Add a New Column to a DataFrame

Aim:

To add a new column to an existing DataFrame using assignment and computed values.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Name': ['Aman', 'Riya'], 'Marks1': [80,
90], 'Marks2': [70, 85]})
df['Total'] = df['Marks1'] + df['Marks2']
df['Grade'] = ['A', 'A+']
print(df)
```

Output:

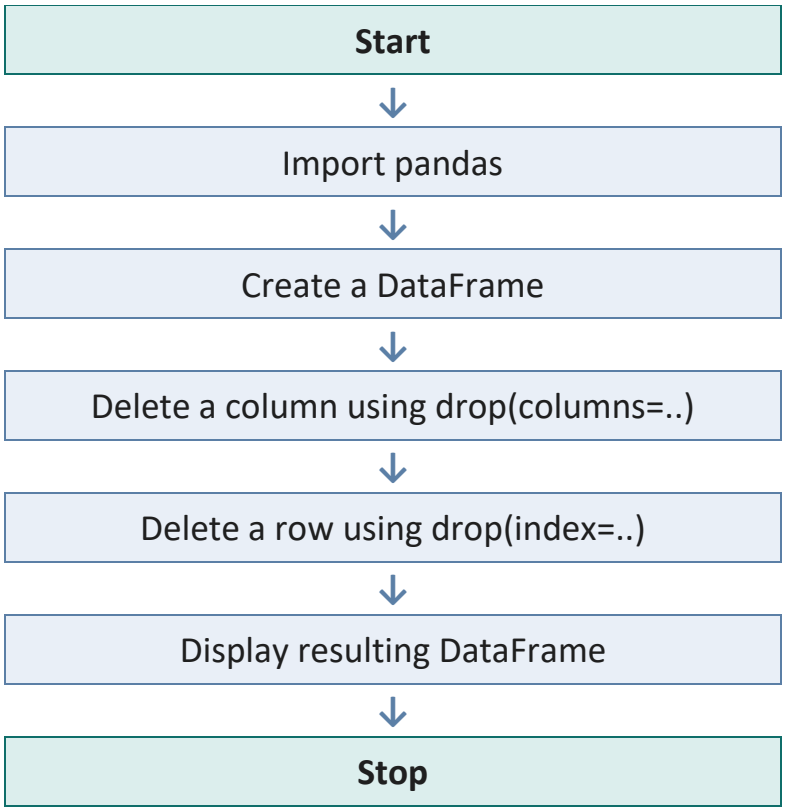
```
   Name  Marks1  Marks2  Total Grade
0  Aman      80      70    150     A
1  Riya      90      85    175    A+
```

Program 13: Delete a Column/Row from a DataFrame

Aim:

To delete columns and rows from a DataFrame using the drop() function.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Name': ['A','B','C'], 'Age': [17,18,17],
'City':['Delhi','Pune','Agra']})
df1 = df.drop(columns=['City'])
df2 = df.drop(index=1)
print(df1)
print(df2)
```

Output:

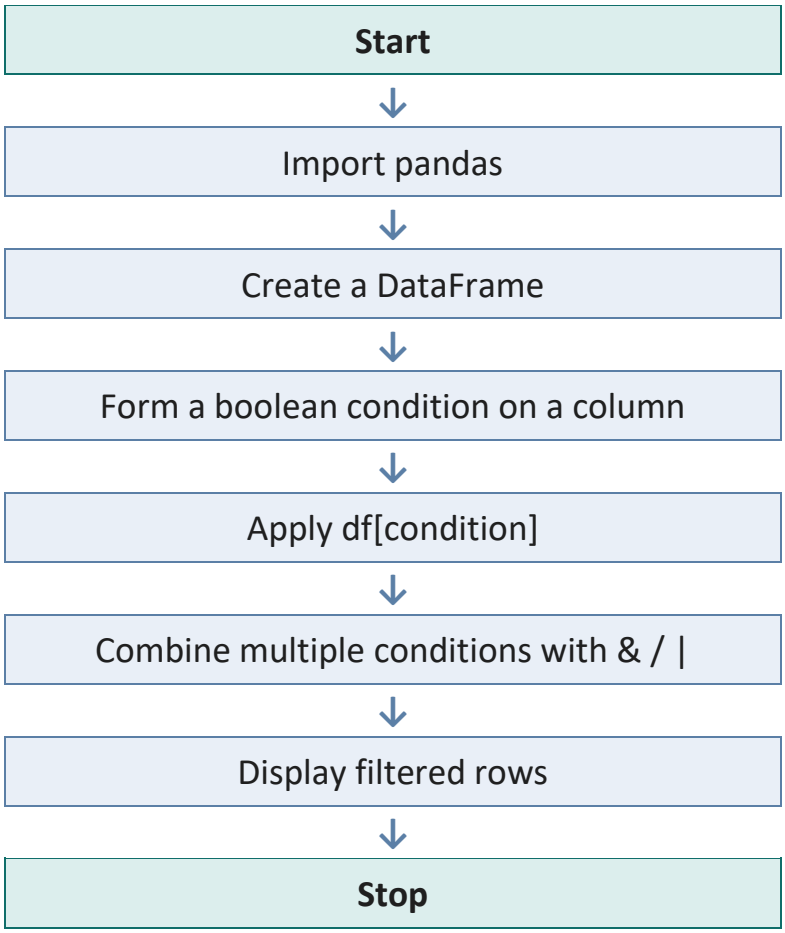
```
  Name  Age
0    A   17
1    B   18
2    C   17
  Name  Age   City
0    A   17  Delhi
2    C   17   Agra
```

Program 14: Filter Rows using Boolean Conditions

Aim:

To select rows from a DataFrame that satisfy one or more given conditions.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Name':['A','B','C','D'],
'Marks':[45,89,60,92], 'Stream':['Sci','Com','Sci','Com']})
print(df[df['Marks'] > 60])
print(df[(df['Marks'] > 60) & (df['Stream'] == 'Com')])
```

Output:

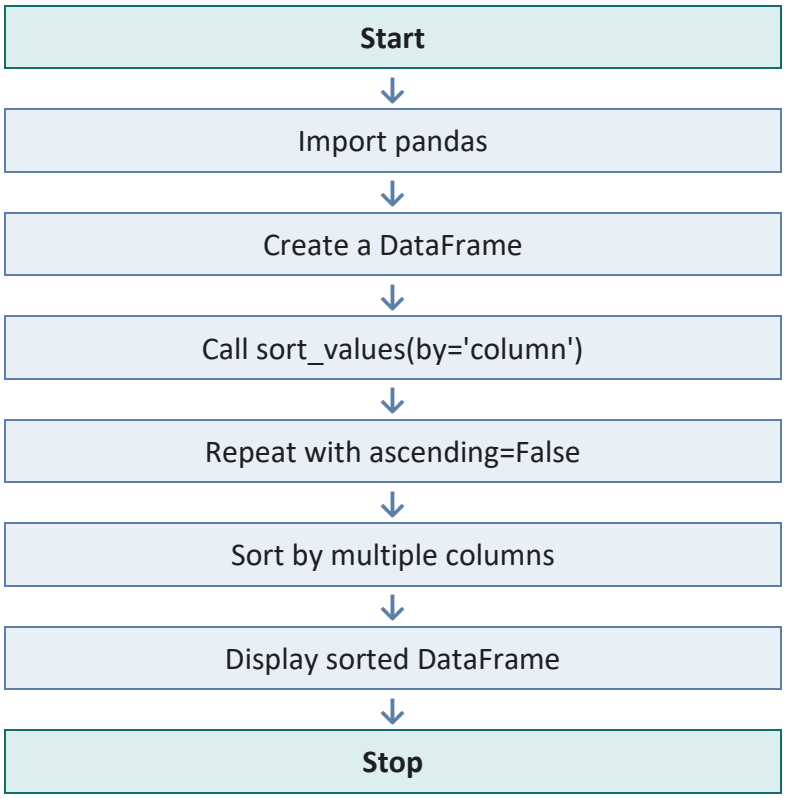
```
  Name  Marks Stream
1    B     89    Com
3    D     92    Com
  Name  Marks Stream
1    B     89    Com
3    D     92    Com
```

Program 15: Sort a DataFrame using sort_values()

Aim:

To sort a DataFrame by one or more columns in ascending or descending order.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Name':['C','A','B'], 'Marks':[70,95,85]})
print(df.sort_values(by='Marks'))
print(df.sort_values(by='Marks', ascending=False))
print(df.sort_values(by=['Marks','Name']))
```

Output:

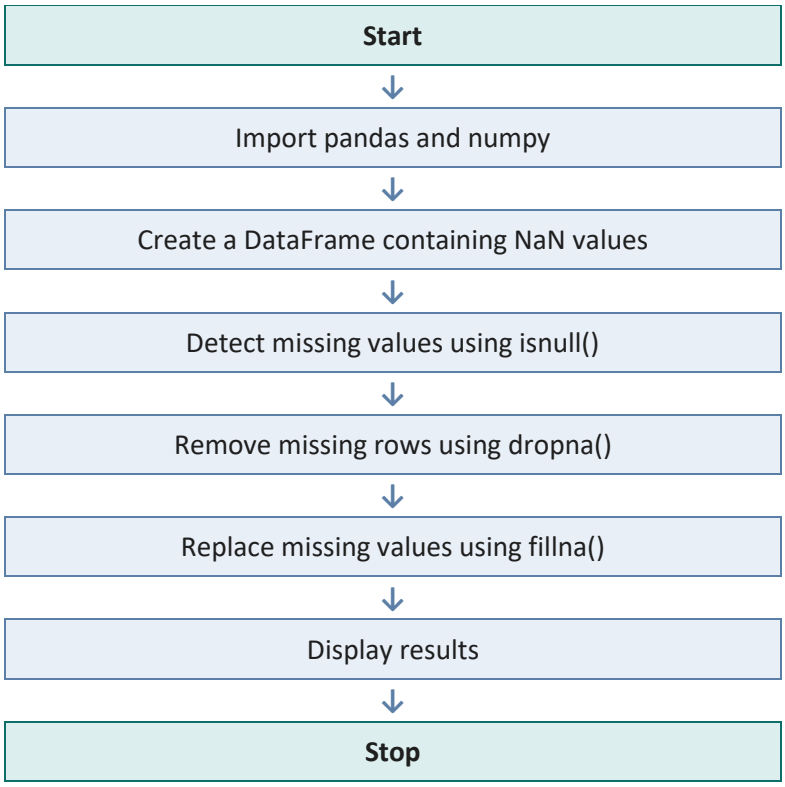
```
  Name  Marks
0    C     70
2    B     85
1    A     95
  Name  Marks
1    A     95
2    B     85
0    C     70
  Name  Marks
0    C     70
2    B     85
1    A     95
```

Program 16: Handle Missing Data (isnull, dropna, fillna)

Aim:

To detect and handle missing values (NaN) in a DataFrame.

Flow Chart:



Program (Source Code):

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Name':['A','B','C'], 'Marks':[78, np.nan,
90]})
print(df.isnull())
print(df.dropna())
print(df.fillna(0))
```

Output:

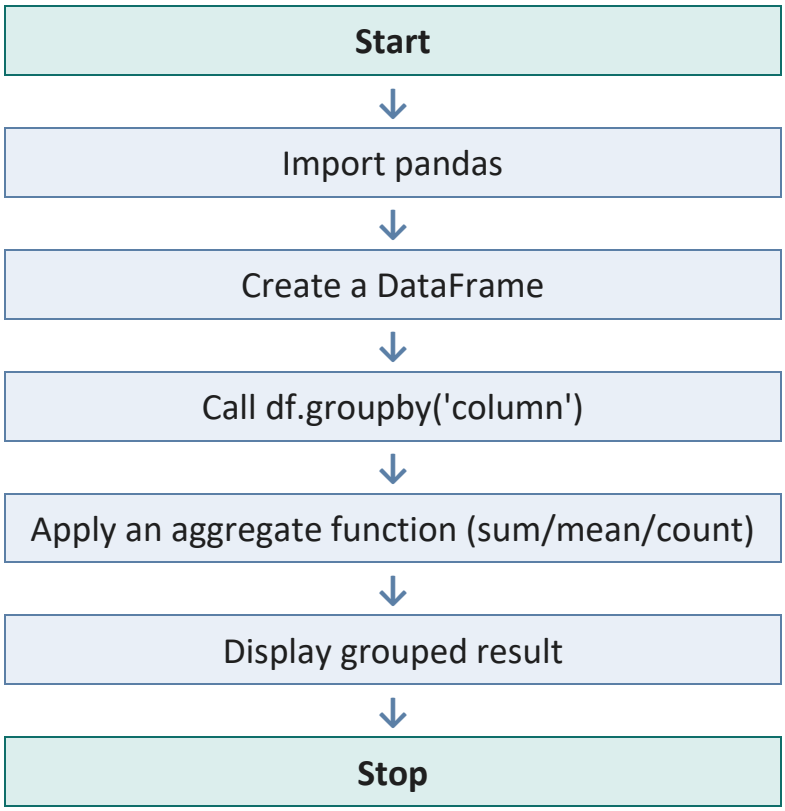
```
    Name  Marks
0  False  False
1  False   True
2  False  False
  Name  Marks
0    A   78.0
2    C   90.0
  Name  Marks
0    A   78.0
1    B    0.0
2    C   90.0
```

Program 17: Group Data using groupby() and Aggregate

Aim:

To group DataFrame rows on a column and apply an aggregate function on each group.

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'Dept':['Sci','Com','Sci','Com'],
'Sales':[200,150,300,180]})
print(df.groupby('Dept')['Sales'].sum())
print(df.groupby('Dept')['Sales'].mean())
```

Output:

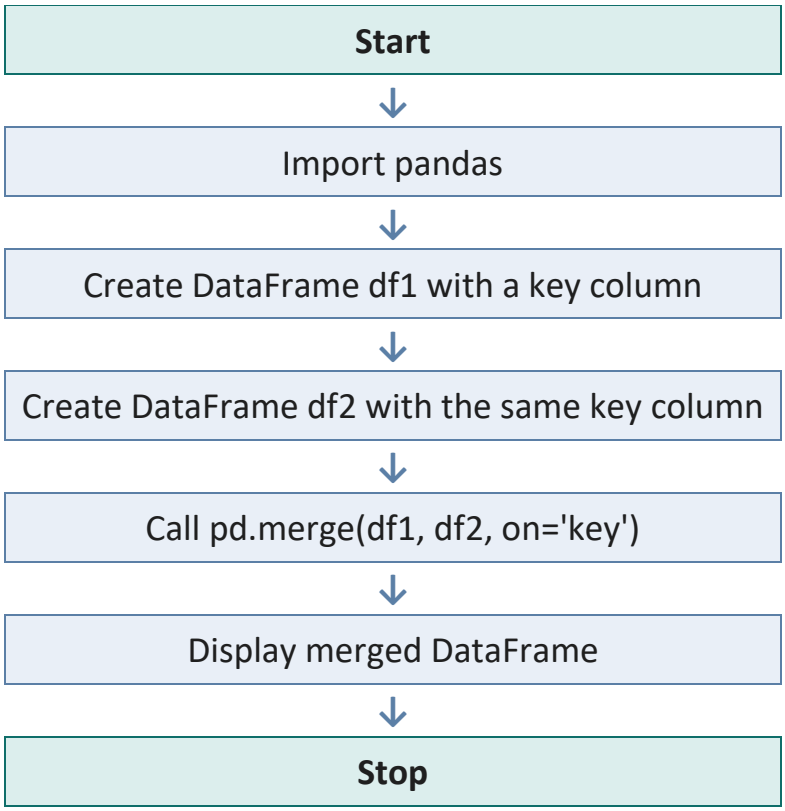
```
Dept
Com    330
Sci    500
Name: Sales, dtype: int64
Dept
Com    165.0
Sci    250.0
Name: Sales, dtype: float64
```

Program 18: Merge Two DataFrames

Aim:

To combine two DataFrames on a common column using pd.merge().

Flow Chart:



Program (Source Code):

```
import pandas as pd

df1 = pd.DataFrame({'RollNo':[1,2,3], 'Name':['A','B','C']})
df2 = pd.DataFrame({'RollNo':[1,2,3], 'Marks':[88,76,90]})
merged = pd.merge(df1, df2, on='RollNo')
print(merged)
```

Output:

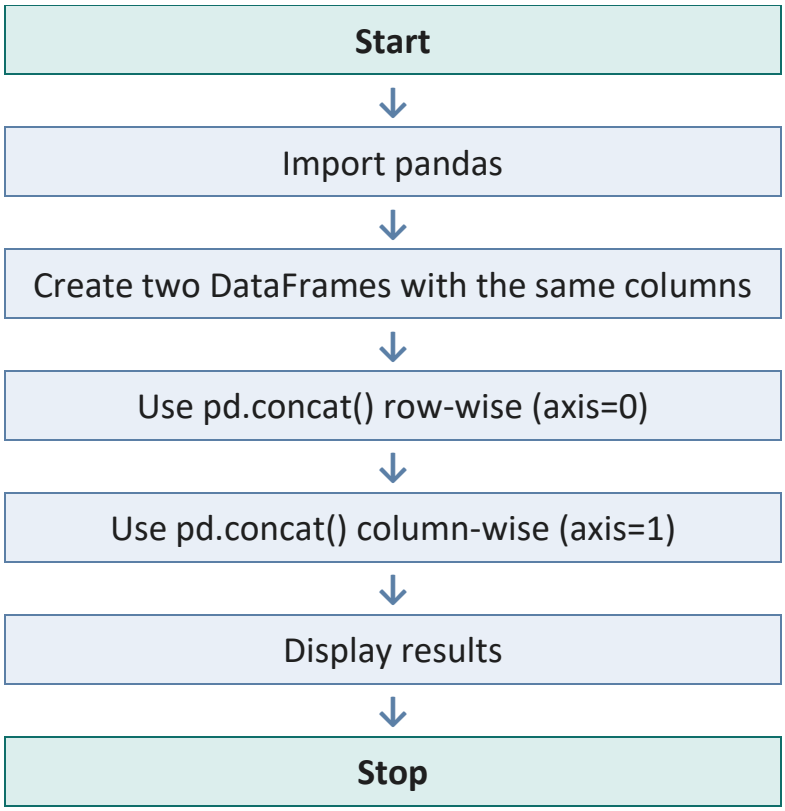
```
   RollNo Name  Marks
0       1    A     88
1       2    B     76
2       3    C     90
```

Program 19: Concatenate Two DataFrames

Aim:

To combine two DataFrames row-wise and column-wise using pd.concat().

Flow Chart:



Program (Source Code):

```
import pandas as pd

df1 = pd.DataFrame({'Name':['A','B']})
df2 = pd.DataFrame({'Name':['C','D']})
print(pd.concat([df1, df2], ignore_index=True))
print(pd.concat([df1, df2], axis=1))
```

Output:

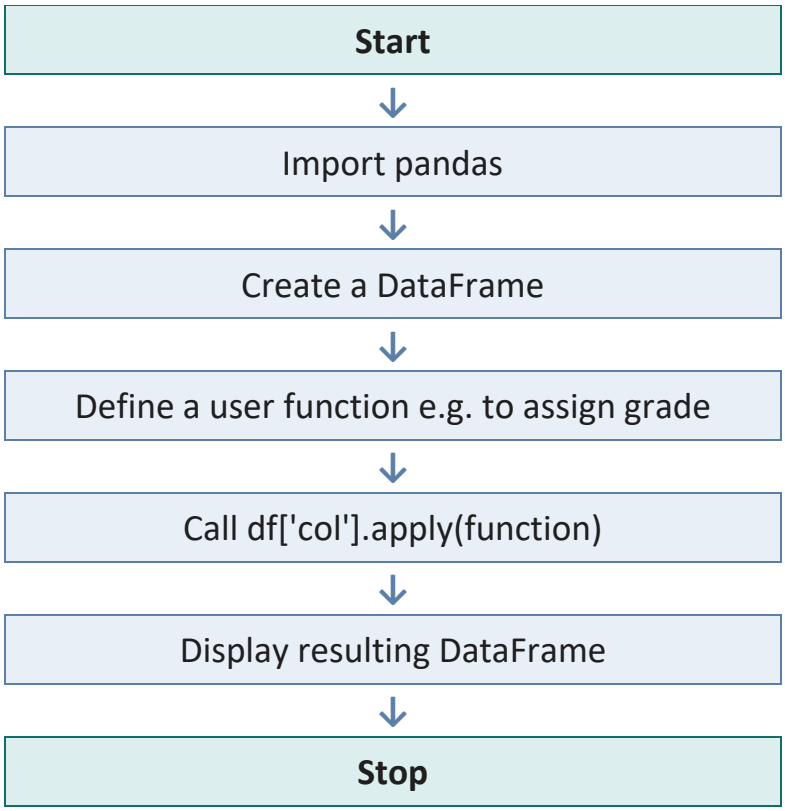
```
  Name
0    A
1    B
2    C
3    D
  Name Name
0    A    C
1    B    D
```

Program 20: Apply a User-Defined Function using apply()

Aim:

To apply a custom function to each element/column of a DataFrame using apply().

Flow Chart:



Program (Source Code):

```
import pandas as pd

def grade(m):
    return 'A' if m>=80 else ('B' if m>=60 else 'C')

df = pd.DataFrame({'Name':['A','B','C'], 'Marks':[85,65,45]})
df['Grade'] = df['Marks'].apply(grade)
print(df)
```

Output:

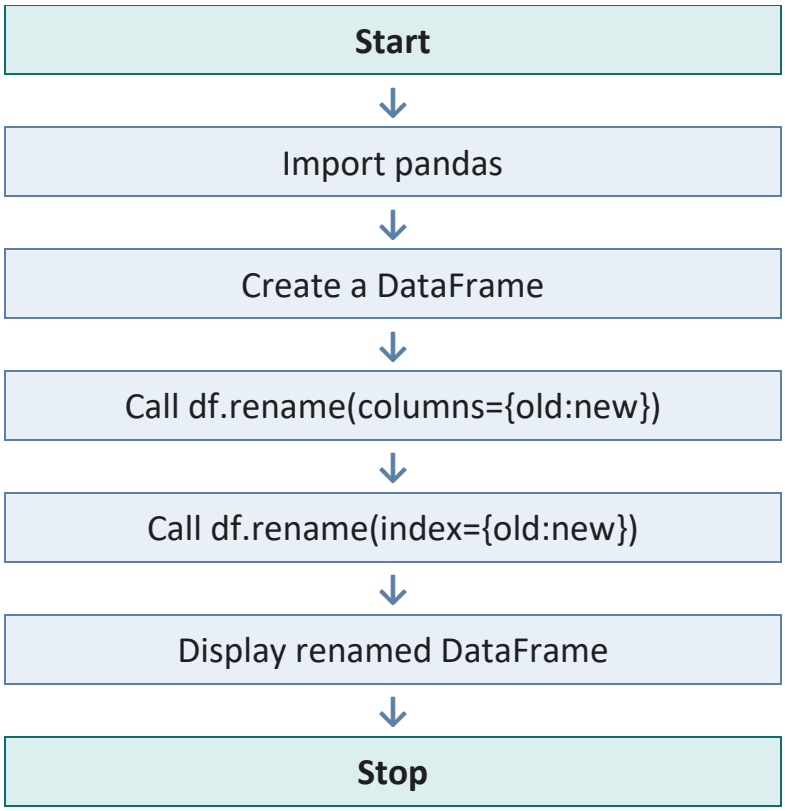
```
  Name  Marks Grade
0    A     85     A
1    B     65     B
2    C     45     C
```

Program 21: Rename Columns and Index of a DataFrame

Aim:

To rename column names and row index labels of a DataFrame using rename().

Flow Chart:



Program (Source Code):

```
import pandas as pd

df = pd.DataFrame({'nm':['A','B'], 'mk':[80,90]})
df = df.rename(columns={'nm':'Name', 'mk':'Marks'})
df = df.rename(index={0:'r1', 1:'r2'})
print(df)
```

Output:

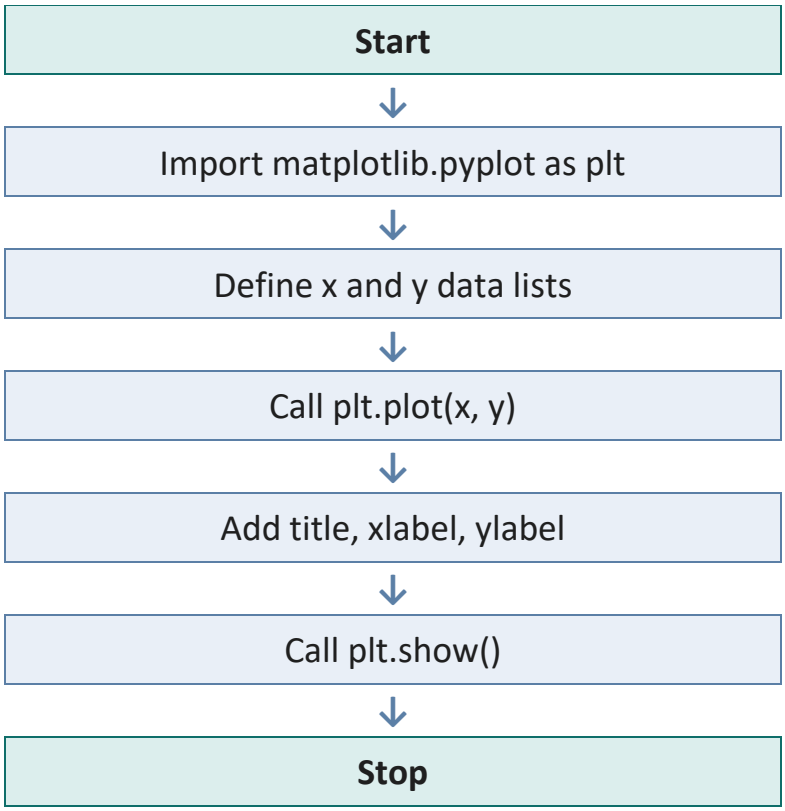
```
   Name  Marks
r1    A     80
r2    B     90
```

Program 22: Plot a Line Chart using Matplotlib

Aim:

To visualize trend data using a line chart with plt.plot().

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt

months = ['Jan','Feb','Mar','Apr','May']
sales = [200, 240, 260, 300, 280]
plt.plot(months, sales, marker='o', color='blue')
plt.title("Monthly Sales Trend")
plt.xlabel("Month")
plt.ylabel("Sales (in units)")
plt.show()
```

Output:

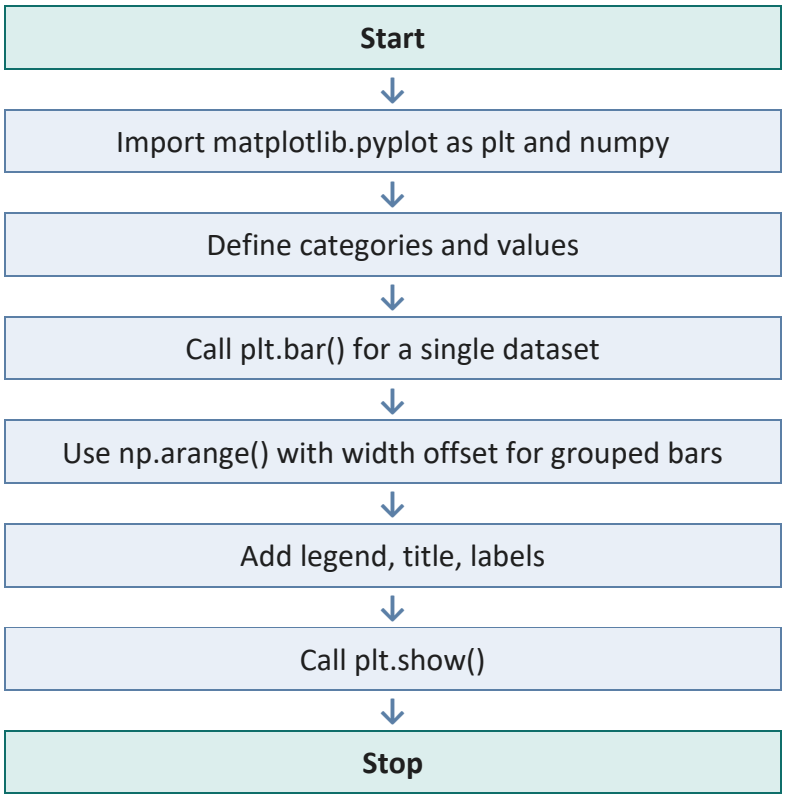
```
A line graph window opens showing sales rising and
dipping across Jan-May, with a titled chart, labelled
axes and circular markers on each data point.
```

Program 23: Plot a Bar Chart and a Multiple Bar Chart

Aim:

To compare categorical data using a simple bar chart and a grouped (multiple) bar chart.

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt
import numpy as np

subjects = ['Eng','Maths','IP']
term1 = [78, 88, 91]
term2 = [82, 90, 95]
x = np.arange(len(subjects))
w = 0.35
plt.bar(x - w/2, term1, width=w, label='Term1')
plt.bar(x + w/2, term2, width=w, label='Term2')
plt.xticks(x, subjects)
plt.title("Term-wise Marks Comparison")
plt.legend()
plt.show()
```

Output:

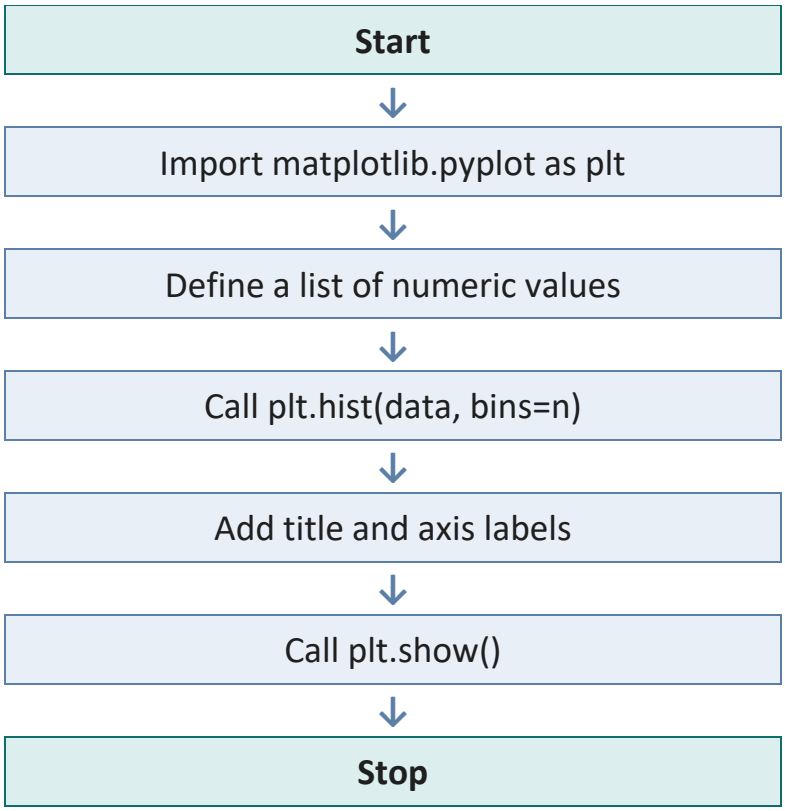
```
A bar chart window opens with two side-by-side bars
per subject (Term1 in one colour, Term2 in another),
a legend box, and subject names on the x-axis.
```

Program 24: Plot a Histogram

Aim:

To study the frequency distribution of numeric data using plt.hist().

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt

marks = [45,67,88,90,55,60,72,95,40,85,77,62]
plt.hist(marks, bins=5, color='skyblue', edgecolor='black')
plt.title("Distribution of Marks")
plt.xlabel("Marks")
plt.ylabel("Number of Students")
plt.show()
```

Output:

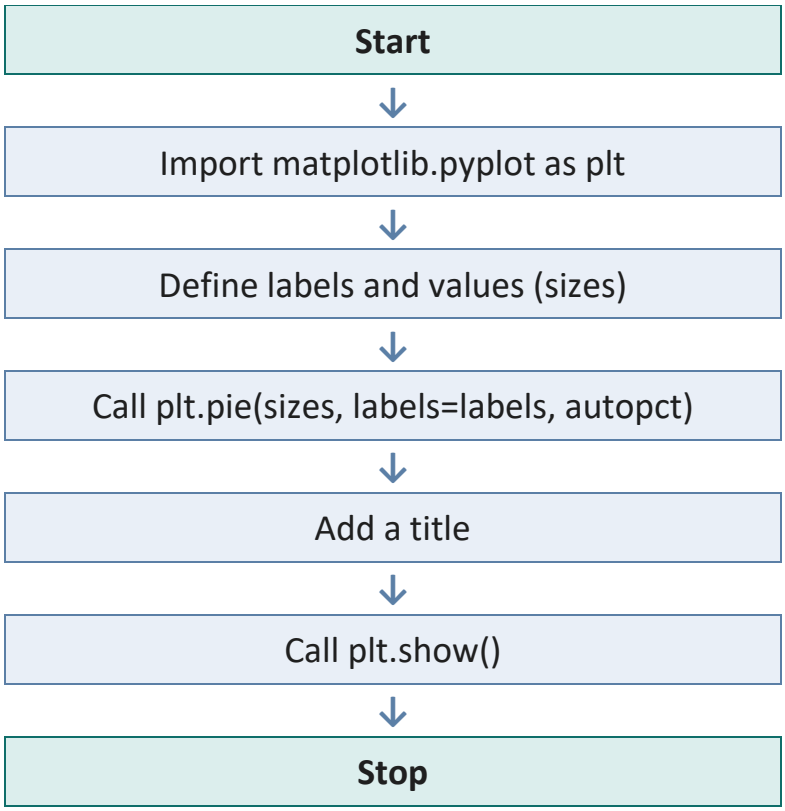
```
A histogram window opens showing 5 bins across the
marks range with bar heights representing the count
of students falling in each mark interval.
```

Program 25: Plot a Pie Chart

Aim:

To represent proportional data using a pie chart with percentage labels.

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt

streams = ['Science','Commerce','Humanities']
students = [120, 90, 60]
plt.pie(students, labels=streams, autopct='%1.1f%%',
startangle=90)
plt.title("Stream-wise Student Distribution")
plt.show()
```

Output:

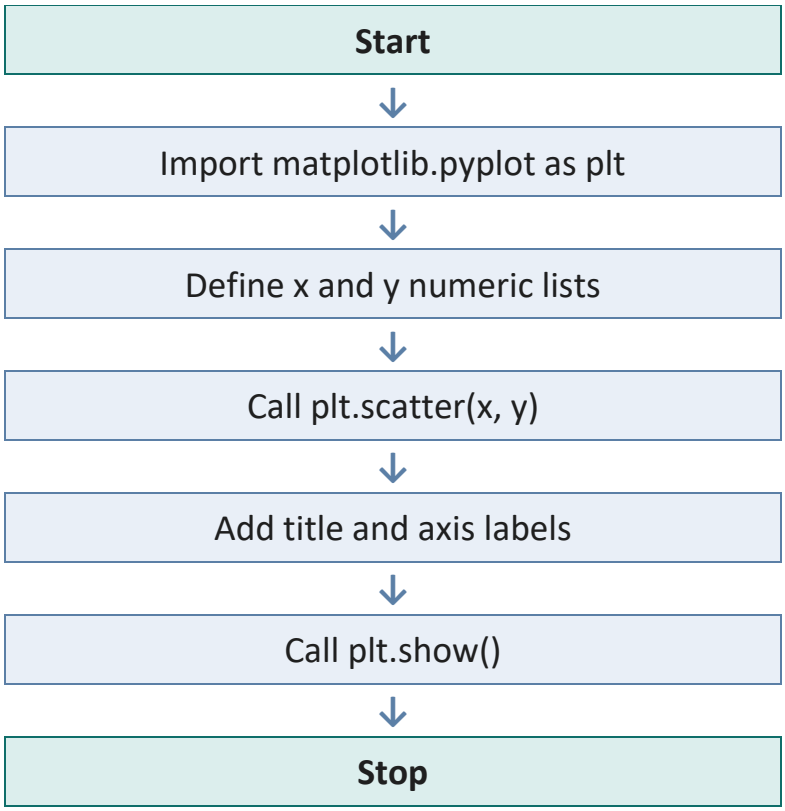
```
A pie chart window opens divided into three slices
(Science/Commerce/Humanities) each labelled with its
percentage share of the total students.
```

Program 26: Plot a Scatter Chart

Aim:

To study the relationship between two numeric variables using a scatter plot.

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt

study_hrs = [1,2,3,4,5,6,7]
marks = [35,42,50,58,66,80,92]
plt.scatter(study_hrs, marks, color='green')
plt.title("Study Hours vs Marks")
plt.xlabel("Study Hours")
plt.ylabel("Marks Obtained")
plt.show()
```

Output:

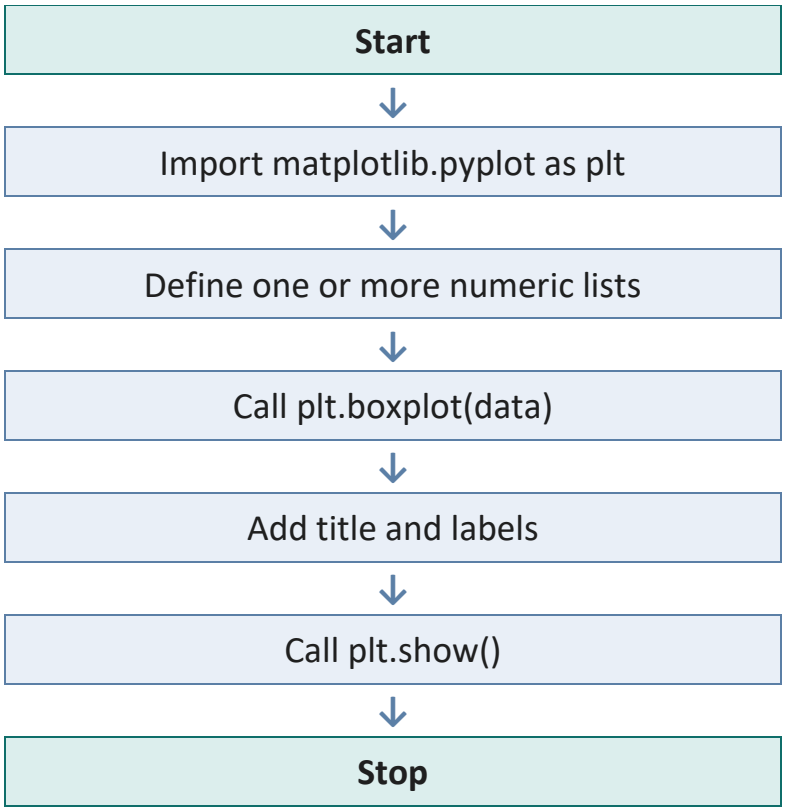
```
A scatter plot window opens showing individual points
that trend upward, indicating marks increase broadly
with study hours.
```

Program 27: Plot a Box Plot

Aim:

To study the spread, median and outliers of a numeric dataset using a box plot.

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt

sec_A = [45,55,60,65,70,72,90,40]
sec_B = [50,60,62,68,74,78,85,95]
plt.boxplot([sec_A, sec_B], labels=['Section A','Section B'])
plt.title("Marks Spread: Section A vs Section B")
plt.ylabel("Marks")
plt.show()
```

Output:

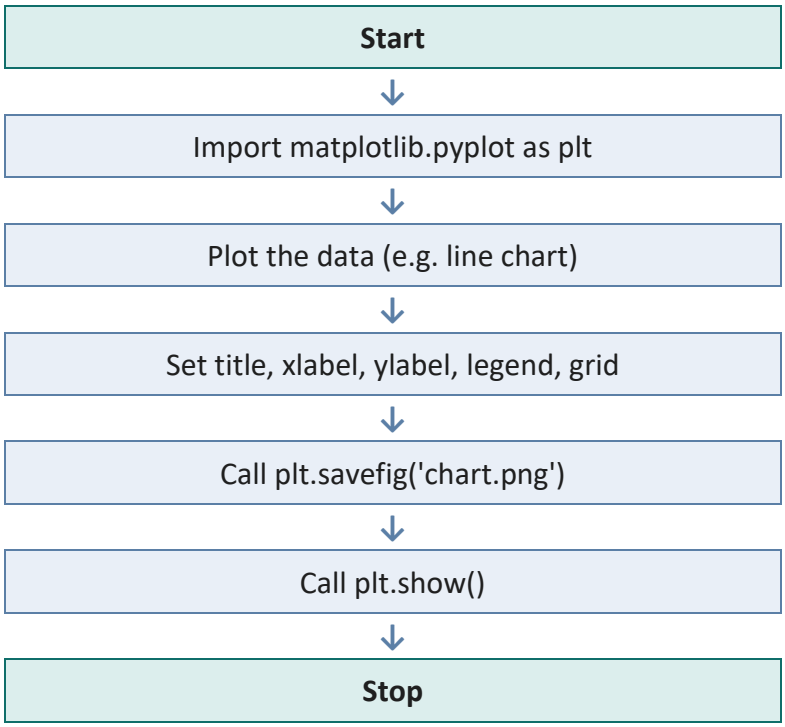
```
A box plot window opens with two boxes (one per
section) showing median line, quartile box, whiskers,
and any outlier points as separate dots.
```

Program 28: Customize a Chart (Title, Labels, Legend, Grid, Save)

Aim:

To apply complete customization to a chart and save it as an image file using savefig().

Flow Chart:



Program (Source Code):

```
import matplotlib.pyplot as plt

days = ['Mon','Tue','Wed','Thu','Fri']
temp = [30,32,29,31,33]
plt.plot(days, temp, color='red', linestyle='--', marker='s',
label='Temperature')
plt.title("Weekly Temperature", fontsize=14)
plt.xlabel("Day")
plt.ylabel("Temp (°C)")
plt.legend()
plt.grid(True)
plt.savefig('weekly_temp.png')
plt.show()
```

Output:

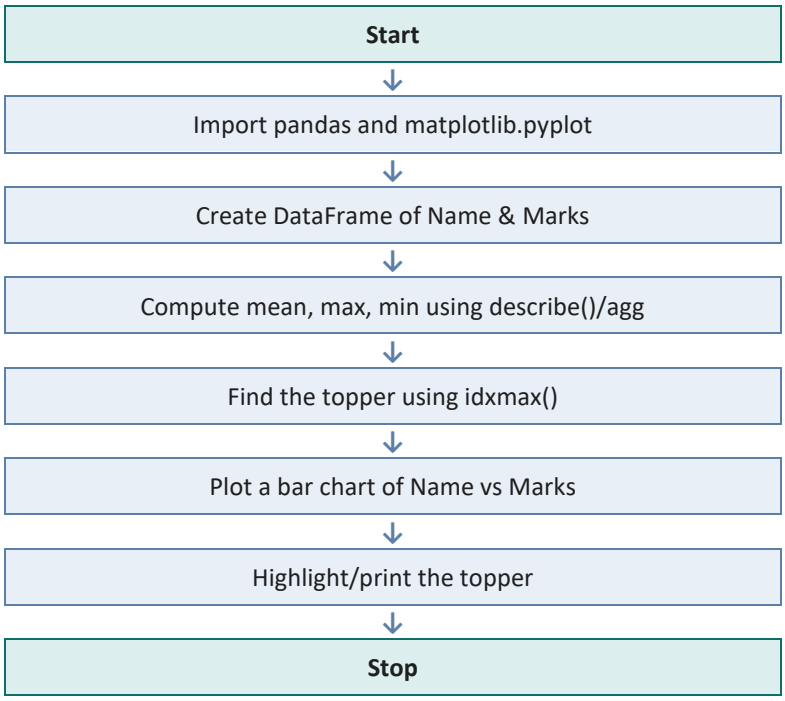
```
A fully labelled, dashed red line chart with square
markers, a legend, and a background grid is displayed
and also saved as 'weekly_temp.png' in the working
folder.
```

Program 29: Mini Project: Analyse a Student Marks DataFrame with Statistics and Chart

Aim:

To build a DataFrame of student marks, compute summary statistics, find the topper, and visualize results with a bar chart.

Flow Chart:



Program (Source Code):

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.DataFrame({'Name':['Aman','Riya','Karan','Simran'],
                   'Marks':[78,92,65,88]})
print("Average Marks:", df['Marks'].mean())
print("Highest Marks:", df['Marks'].max())
topper = df.loc[df['Marks'].idxmax(), 'Name']
print("Topper:", topper)

plt.bar(df['Name'], df['Marks'], color='orange')
plt.title("Student Marks")
plt.xlabel("Name")
plt.ylabel("Marks")
plt.show()
```

Output:

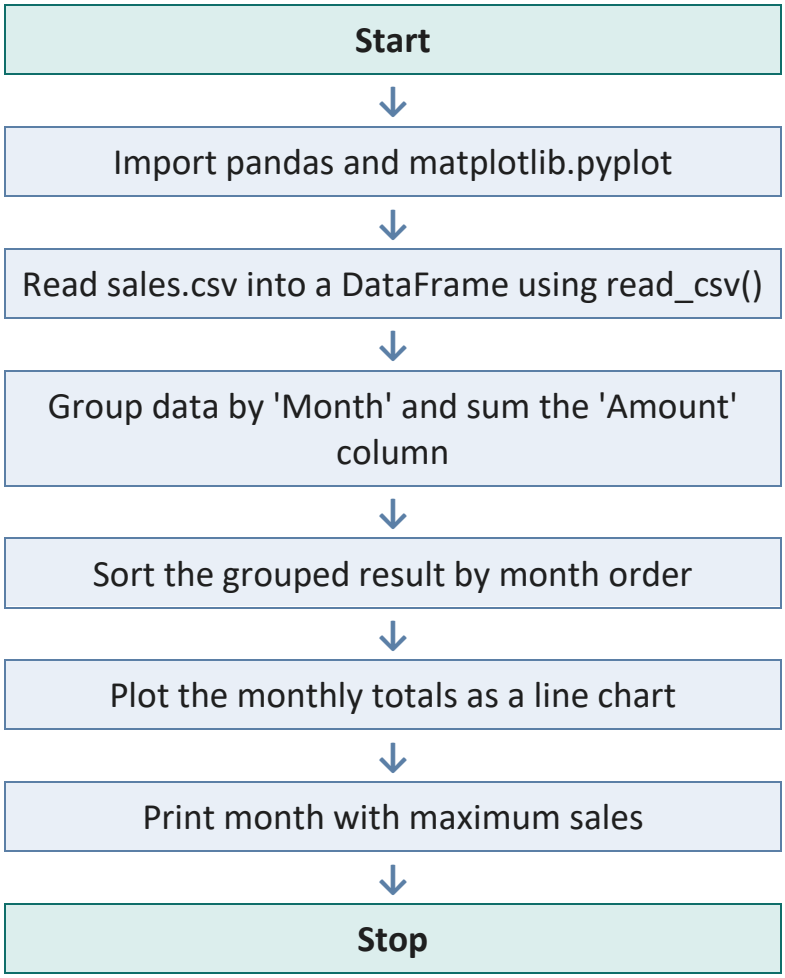
```
Average Marks: 80.75
Highest Marks: 92
Topper: Riya
(A bar chart window also opens showing each student's
marks, with Riya's bar the tallest.)
```

Program 30: Mini Project: Analyse Sales Data from CSV using groupby() and Line Chart

Aim:

To read sales data from a CSV file, summarize monthly totals using groupby(), and plot the trend using a line chart — combining file I/O, DataFrame operations and visualization.

Flow Chart:



Program (Source Code):

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('sales.csv')     # columns: Month, Product,
Amount
monthly = df.groupby('Month')['Amount'].sum()
print(monthly)

best_month = monthly.idxmax()
print("Best Sales Month:", best_month)
```

```
monthly.plot(kind='line', marker='o', color='purple',
title='Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.show()
```

Output:

```
Month
Apr    45200
Feb    38900
Jan    42000
Mar    39750
Name: Amount, dtype: int64
Best Sales Month: Apr
(A purple line chart window opens showing the
month-wise sales trend with markers on each point.)
```

PART B — Python–MySQL Connectivity: Importing & Exporting Data

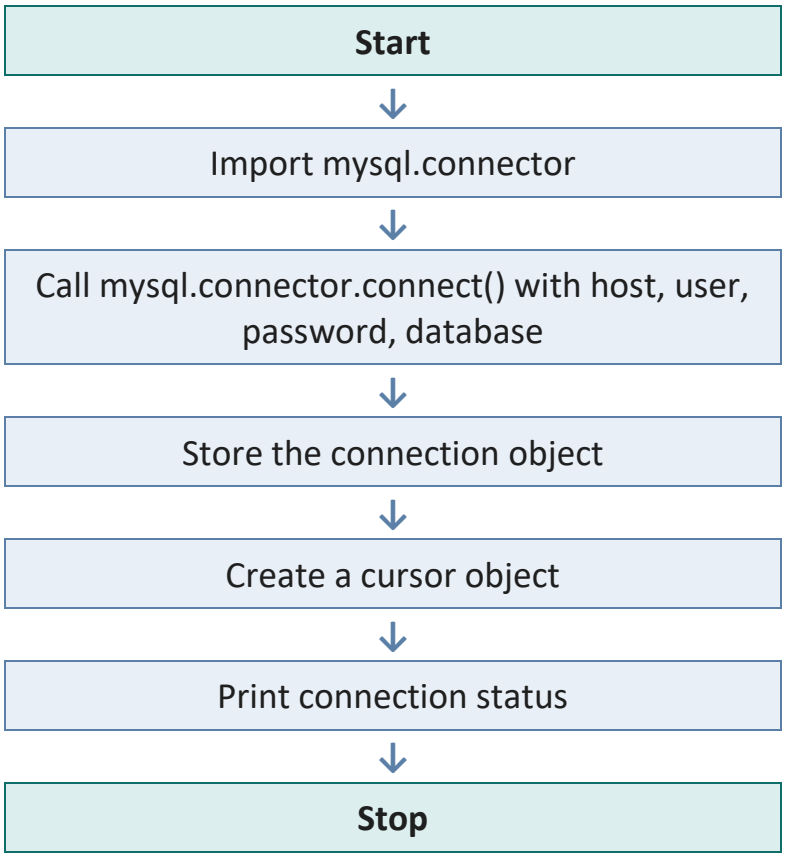
10 Programs covering connecting to MySQL, CRUD operations, and transferring data between Pandas DataFrames and MySQL tables

Program 31: Connect Python to MySQL Database

Aim:

To establish a connection between a Python program and a MySQL database using the mysql.connector module.

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
                  password="root", database="school")
if con.is_connected():
    print("Connected to MySQL successfully")
cursor = con.cursor()
con.close()
```

Output:

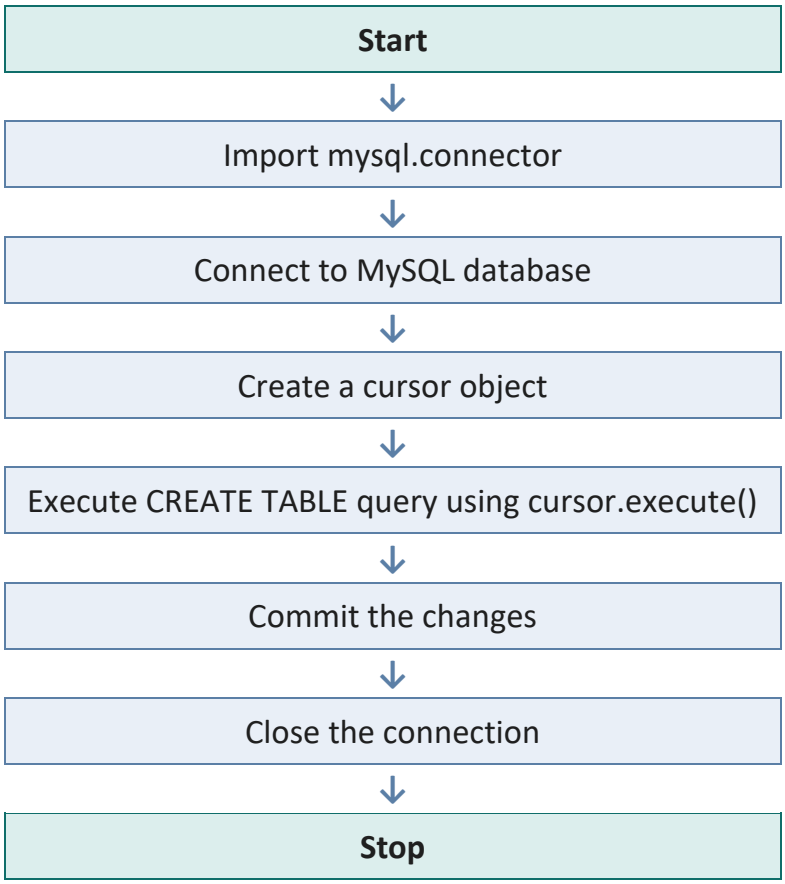
```
Connected to MySQL successfully
```

Program 32: Create a Table in MySQL from Python

Aim:

To create a new table inside a MySQL database by executing a DDL statement from within Python.

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
cursor.execute("""CREATE TABLE IF NOT EXISTS student(
                  rollno INT PRIMARY KEY,
                  name VARCHAR(30),
                  marks INT)""")
con.commit()
print("Table created successfully")
con.close()
```

Output:

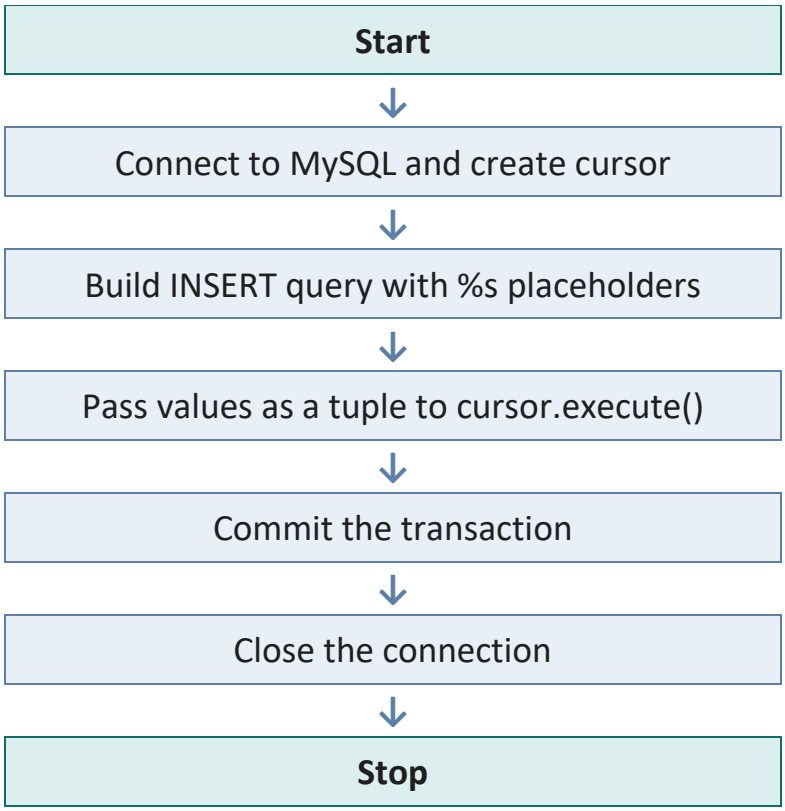
```
Table created successfully
```

Program 33: Insert a Single Record into MySQL from Python

Aim:

To insert one record into a MySQL table using a parameterized query from Python.

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
query = "INSERT INTO student(rollno, name, marks) VALUES (%s,
%s, %s)"
cursor.execute(query, (1, "Aman", 88))
con.commit()
print(cursor.rowcount, "record inserted")
con.close()
```

Output:

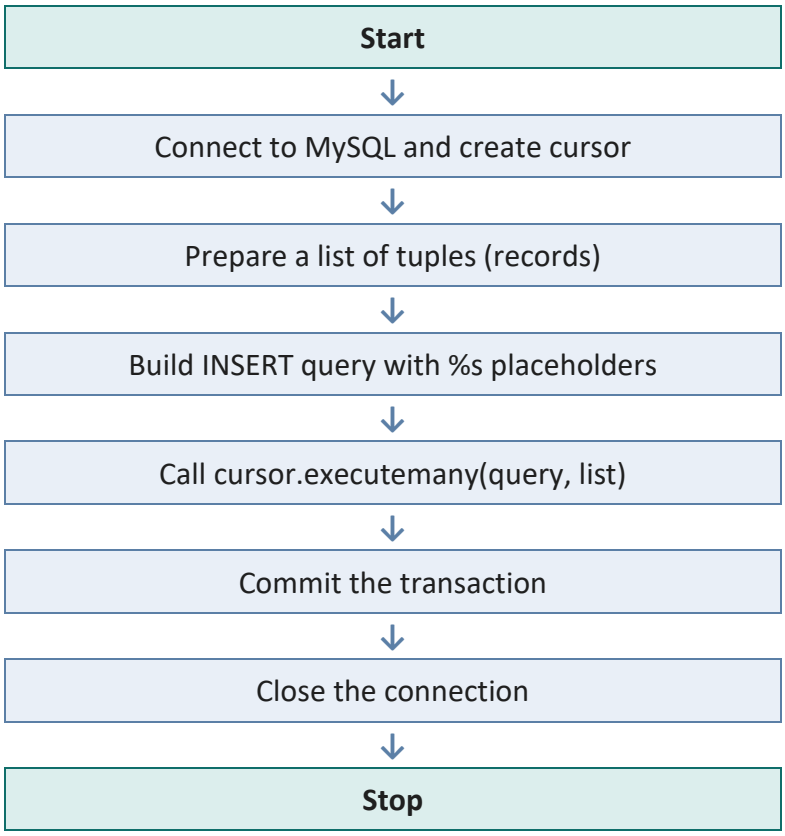
```
1 record inserted
```

Program 34: Insert Multiple Records using executemany()

Aim:

To insert several records into a MySQL table in a single call using cursor.executemany().

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
records = [(2, "Riya", 92), (3, "Karan", 65), (4, "Simran",
78)]
query = "INSERT INTO student(rollno, name, marks) VALUES (%s,
%s, %s)"
cursor.executemany(query, records)
con.commit()
print(cursor.rowcount, "records inserted")
con.close()
```

Output:

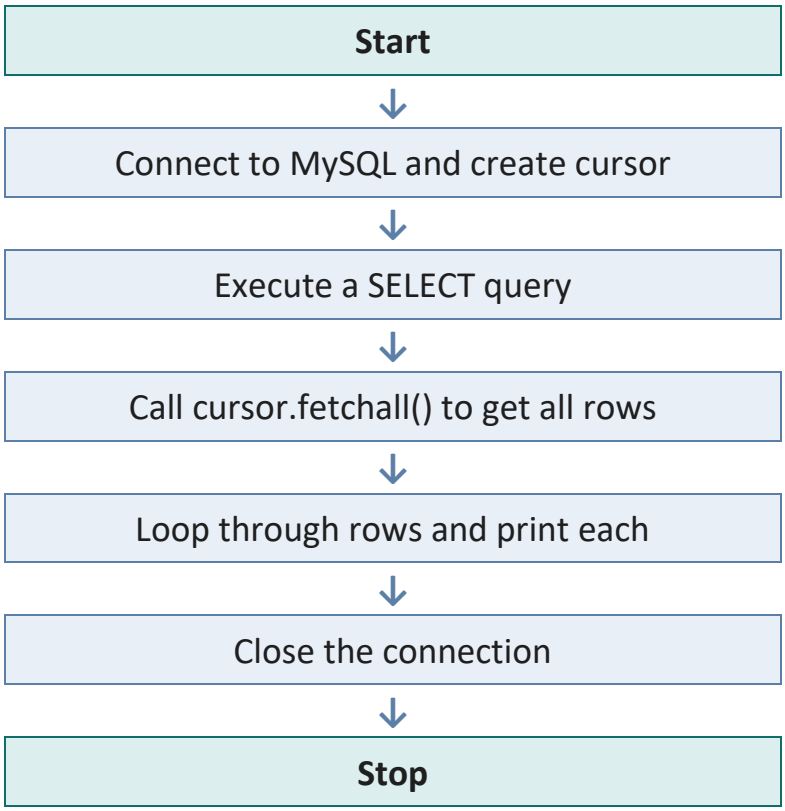
```
3 records inserted
```

Program 35: Fetch and Display Records from MySQL in Python

Aim:

To retrieve all records from a MySQL table into Python using fetchall() and display them.

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
cursor.execute("SELECT * FROM student")
rows = cursor.fetchall()
for row in rows:
    print(row)
con.close()
```

Output:

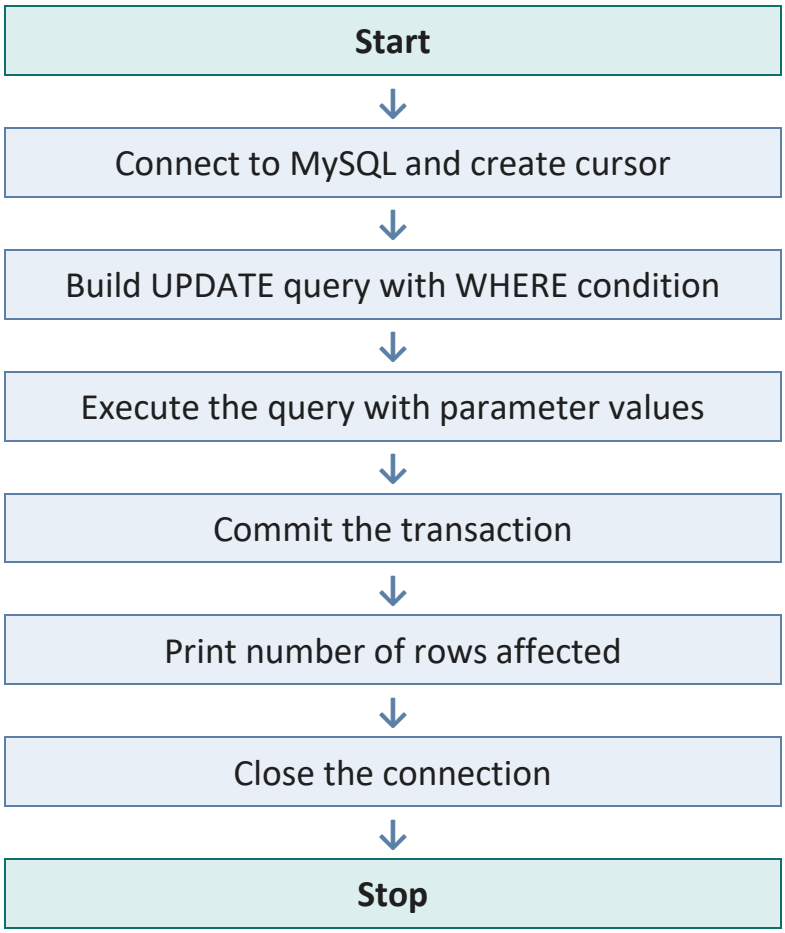
```
(1, 'Aman', 88)
(2, 'Riya', 92)
(3, 'Karan', 65)
(4, 'Simran', 78)
```

Program 36: Update Records in MySQL Table from Python

Aim:

To modify existing records of a MySQL table using an UPDATE query executed from Python.

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
cursor.execute("UPDATE student SET marks = %s WHERE rollno =
%s", (95, 3))
con.commit()
print(cursor.rowcount, "record(s) updated")
con.close()
```

Output:

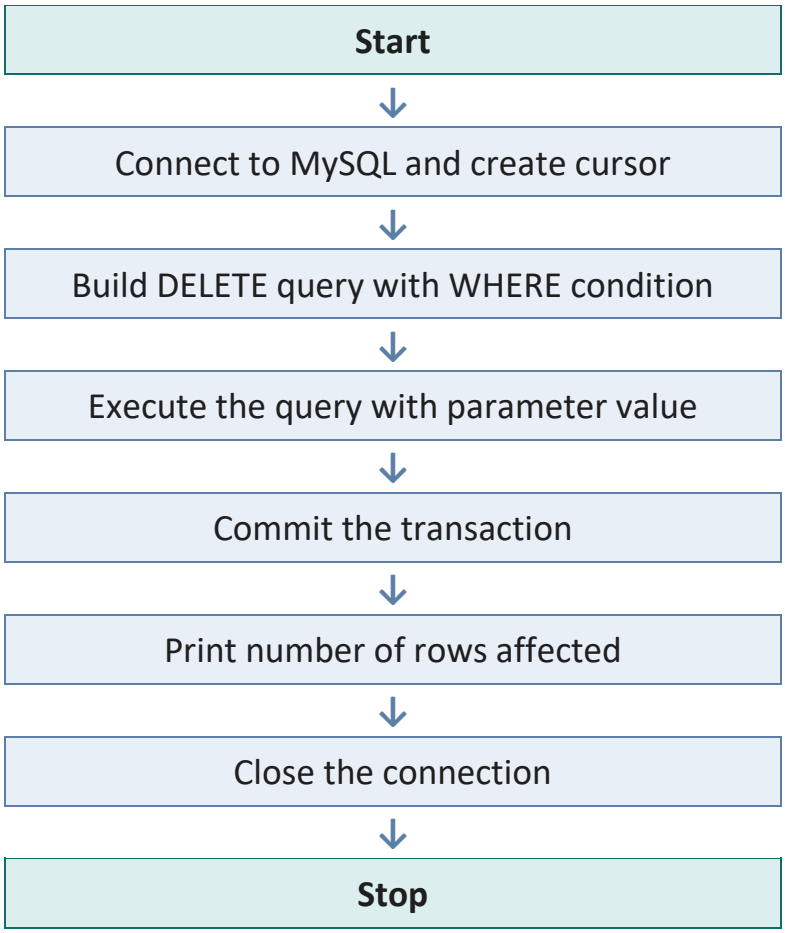
```
1 record(s) updated
```

Program 37: Delete Records from MySQL Table from Python

Aim:

To remove a record from a MySQL table using a DELETE query executed from Python.

Flow Chart:



Program (Source Code):

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
cursor.execute("DELETE FROM student WHERE rollno = %s", (4,))
con.commit()
print(cursor.rowcount, "record(s) deleted")
con.close()
```

Output:

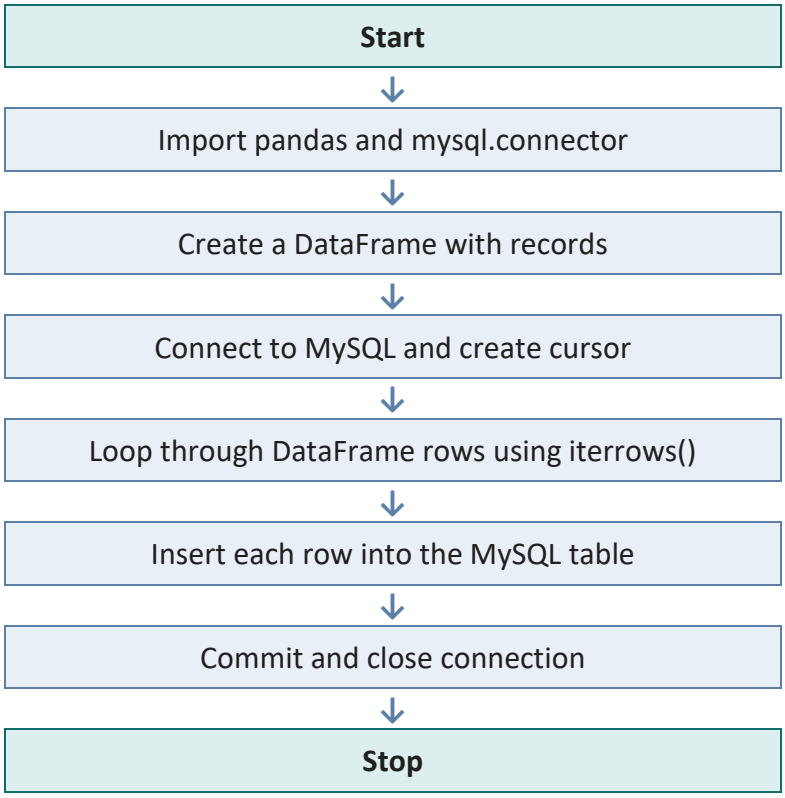
```
1 record(s) deleted
```

Program 38: Export Data from a Pandas DataFrame to a MySQL Table

Aim:

To take data from a Pandas DataFrame in Python and insert it into a MySQL table (Pandas to SQL).

Flow Chart:



Program (Source Code):

```
import pandas as pd
import mysql.connector as sql

df = pd.DataFrame({'rollno':[5,6], 'name':['Ishaan','Meher'],
'marks':[81,73]})

con = sql.connect(host="localhost", user="root",
password="root", database="school")
cursor = con.cursor()
query = "INSERT INTO student(rollno, name, marks) VALUES (%s,
%s, %s)"
for i, row in df.iterrows():
    cursor.execute(query, (row['rollno'], row['name'],
row['marks']))
con.commit()
print("DataFrame exported to MySQL table 'student'")
con.close()
```

Output:

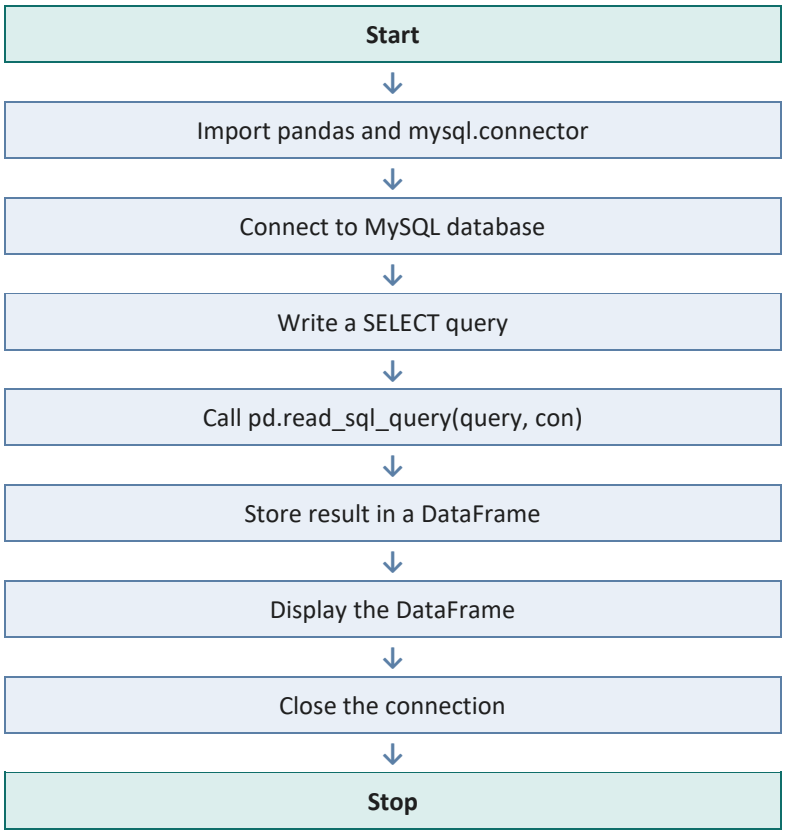
```
DataFrame exported to MySQL table 'student'
```

Program 39: Import Data from a MySQL Table into a Pandas DataFrame

Aim:

To read records from a MySQL table directly into a Pandas DataFrame (SQL to Pandas) using read_sql_query().

Flow Chart:



Program (Source Code):

```
import pandas as pd
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
password="root", database="school")
df = pd.read_sql_query("SELECT * FROM student", con)
print(df)
con.close()
```

Output:

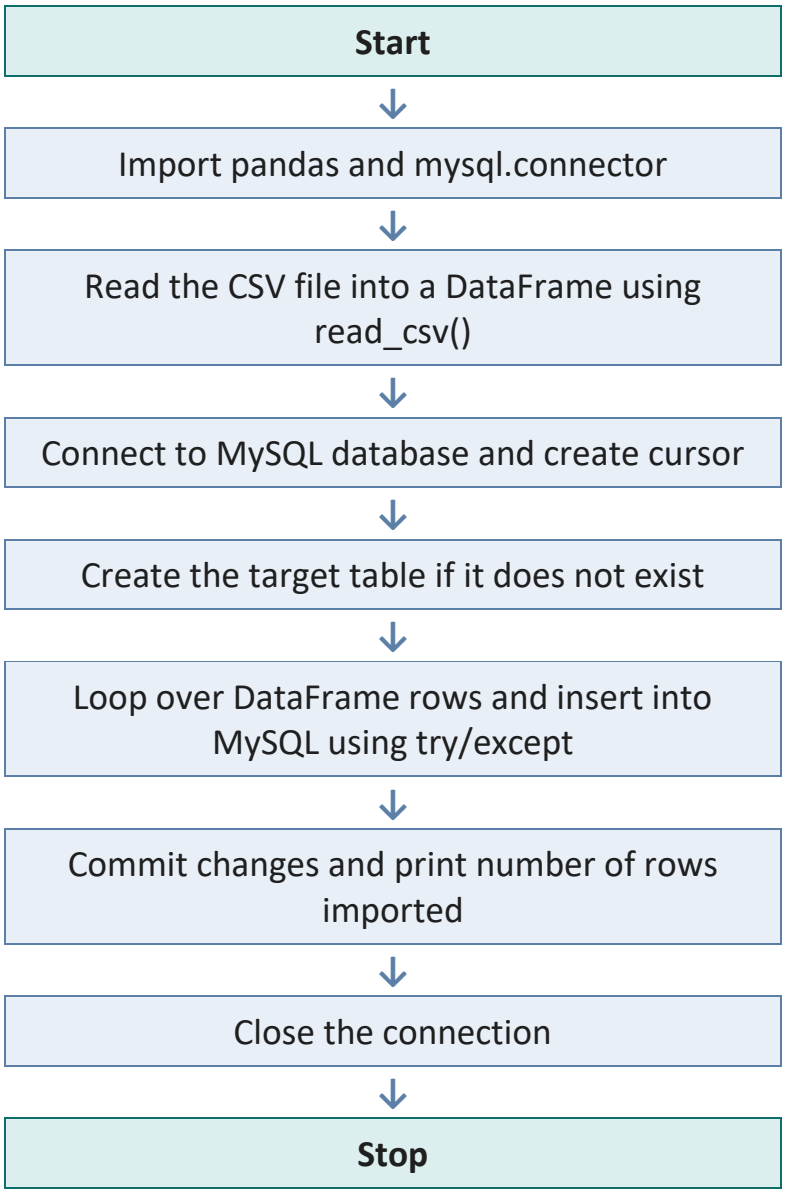
```
   rollno    name  marks
0       1    Aman     88
1       2    Riya     92
2       3   Karan     95
3       5  Ishaan     81
4       6   Meher     73
```

Program 40: Mini Project: Import a CSV File into a MySQL Table using Python

Aim:

To read data from a CSV file using Pandas, and then export it into a MySQL table, combining file handling, DataFrames and MySQL connectivity with basic error handling.

Flow Chart:



Program (Source Code):

```
import pandas as pd
import mysql.connector as sql

df = pd.read_csv('new_students.csv')   # columns:
rollno,name,marks

con = sql.connect(host="localhost", user="root",
password="root", database="school")
```

```
cursor = con.cursor()
cursor.execute("""CREATE TABLE IF NOT EXISTS student(
                    rollno INT PRIMARY KEY, name VARCHAR(30),
marks INT)""")

count = 0
query = "INSERT INTO student(rollno, name, marks) VALUES (%s,
%s, %s)"
for i, row in df.iterrows():
    try:
        cursor.execute(query, (int(row['rollno']),
row['name'], int(row['marks'])))
        count += 1
    except sql.Error as e:
        print("Skipped row", i, "->", e)

con.commit()
print(count, "records imported from CSV to MySQL")
con.close()
```

Output:

```
7 records imported from CSV to MySQL
```

PART C — MySQL Queries and Functions

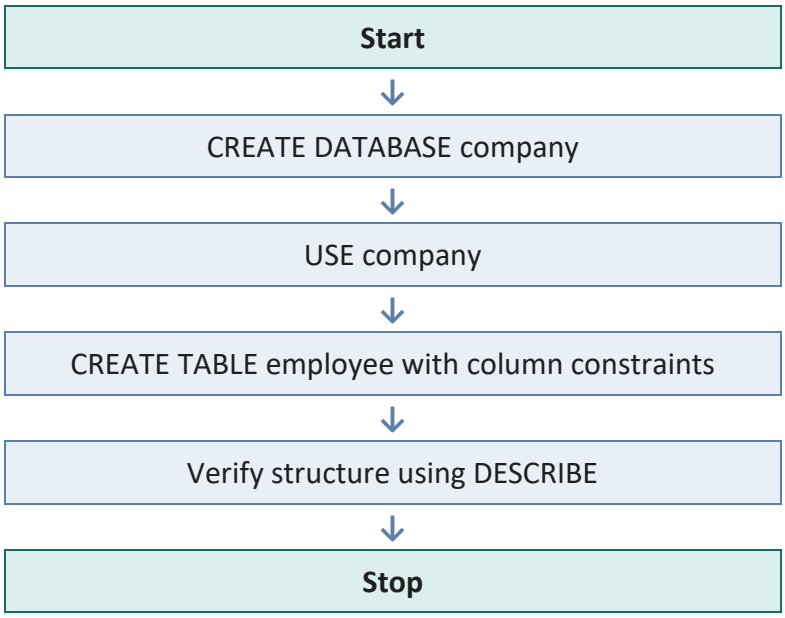
20 Queries covering DDL/DML, constraints, aggregate functions, GROUP BY/HAVING, string/date/math functions, and equi-joins

Query 41: Create Database and Table with Constraints

Aim:

To create a database and a table with PRIMARY KEY and NOT NULL constraints.

Flow Chart:



SQL Statement(s):

```
CREATE DATABASE company;
USE company;

CREATE TABLE employee(
   EmpId INT PRIMARY KEY,
   Ename VARCHAR(30) NOT NULL,
   Dept VARCHAR(20),
   Salary DECIMAL(10,2),
   DOJ DATE,
   Gender CHAR(1)
);
DESCRIBE employee;
```

Output:

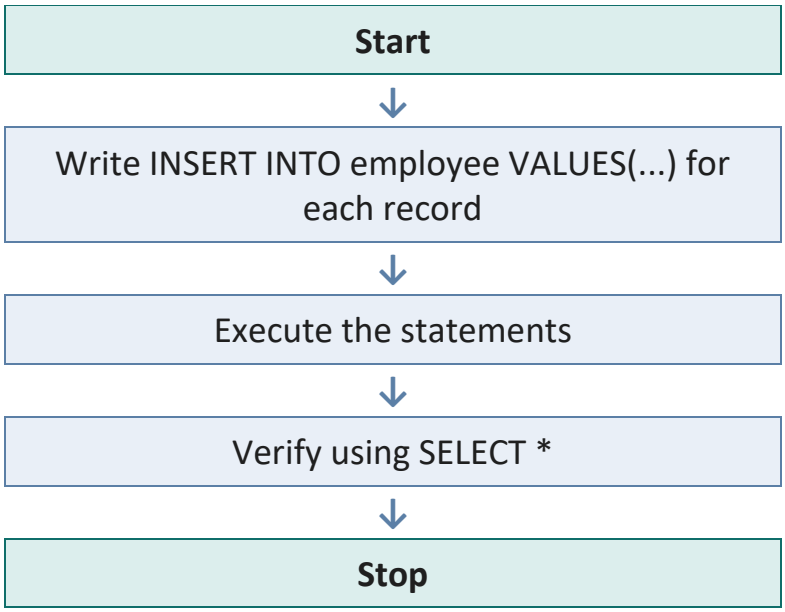
```
Query OK, 1 row affected
Field/Type/Null/Key details of table 'employee'
are displayed.
```

Query 42: Insert Records into the Table

Aim:

To insert multiple records into the employee table using the INSERT command.

Flow Chart:



SQL Statement(s):

```
INSERT INTO employee VALUES(101,'Aman
Sharma','Sales',45000,'2019-06-12','M');
INSERT INTO employee VALUES(102,'Riya
Kapoor','HR',52000,'2020-03-05','F');
INSERT INTO employee VALUES(103,'Karan
Mehta','IT',65000,'2018-11-20','M');
INSERT INTO employee VALUES(104,'Simran
Kaur','IT',72000,'2021-01-15','F');
INSERT INTO employee VALUES(105,'Ishaan
Roy','Sales',38000,'2022-07-01','M');
```

Output:

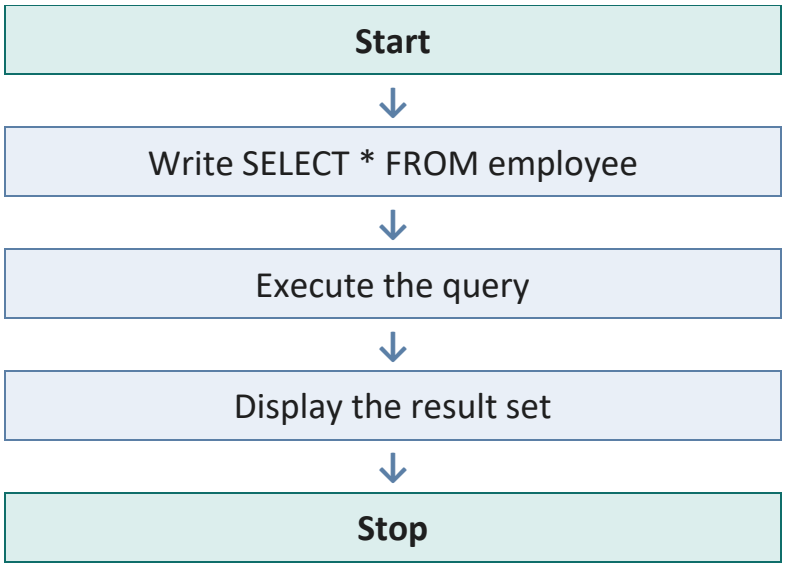
```
5 rows affected (records inserted successfully)
```

Query 43: Display All Records - SELECT *

Aim:

To display all columns and all rows of the employee table.

Flow Chart:



SQL Statement(s):

```
SELECT * FROM employee;
```

Output:

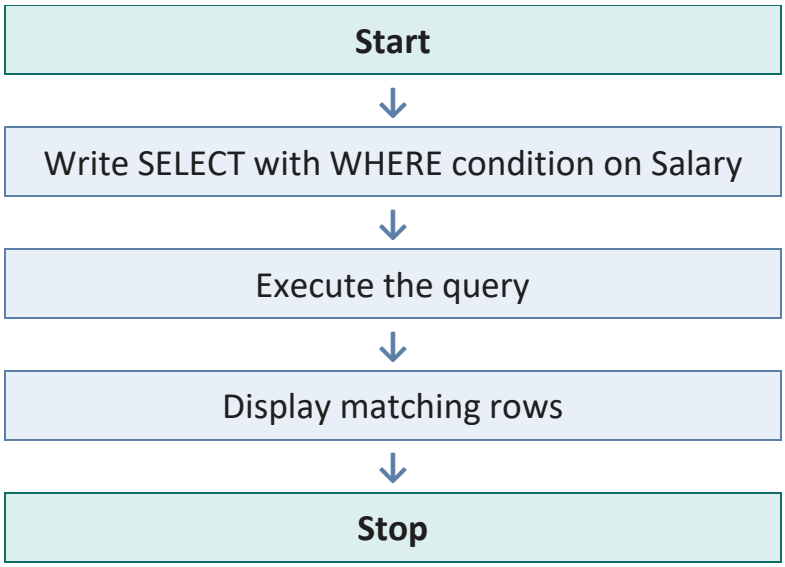
```
EmpId | Ename         | Dept  | Salary | DOJ        | Gender
101   | Aman Sharma   | Sales | 45000  | 2019-06-12 | M
102   | Riya Kapoor   | HR    | 52000  | 2020-03-05 | F
103   | Karan Mehta   | IT    | 65000  | 2018-11-20 | M
104   | Simran Kaur   | IT    | 72000  | 2021-01-15 | F
105   | Ishaan Roy    | Sales | 38000  | 2022-07-01 | M
```

Query 44: SELECT with WHERE Clause

Aim:

To retrieve records that satisfy a specific condition using the WHERE clause.

Flow Chart:



SQL Statement(s):

```
SELECT Ename, Dept, Salary FROM employee
WHERE Salary > 50000;
```

Output:

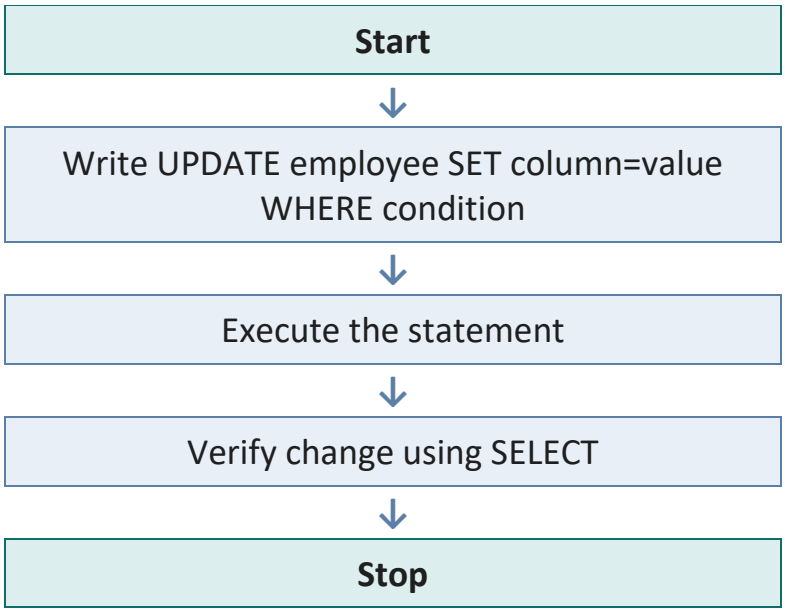
```
Ename         | Dept | Salary
Riya Kapoor   | HR   | 52000
Karan Mehta   | IT   | 65000
Simran Kaur   | IT   | 72000
```

Query 45: UPDATE Command

Aim:

To modify the value of a column for records satisfying a given condition.

Flow Chart:



SQL Statement(s):

```
UPDATE employee SET Salary = Salary + 5000
WHERE Dept = 'IT';
SELECT Ename, Salary FROM employee WHERE Dept='IT';
```

Output:

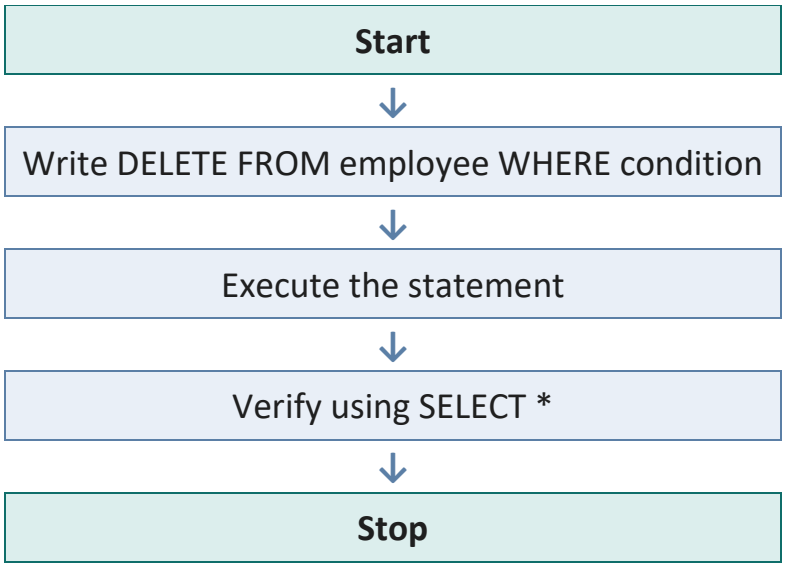
```
Query OK, 2 rows affected
Ename         | Salary
Karan Mehta   | 70000
Simran Kaur   | 77000
```

Query 46: DELETE Command

Aim:

To remove a record from the table based on a condition.

Flow Chart:



SQL Statement(s):

```
DELETE FROM employee WHERE EmpId = 105;
SELECT * FROM employee;
```

Output:

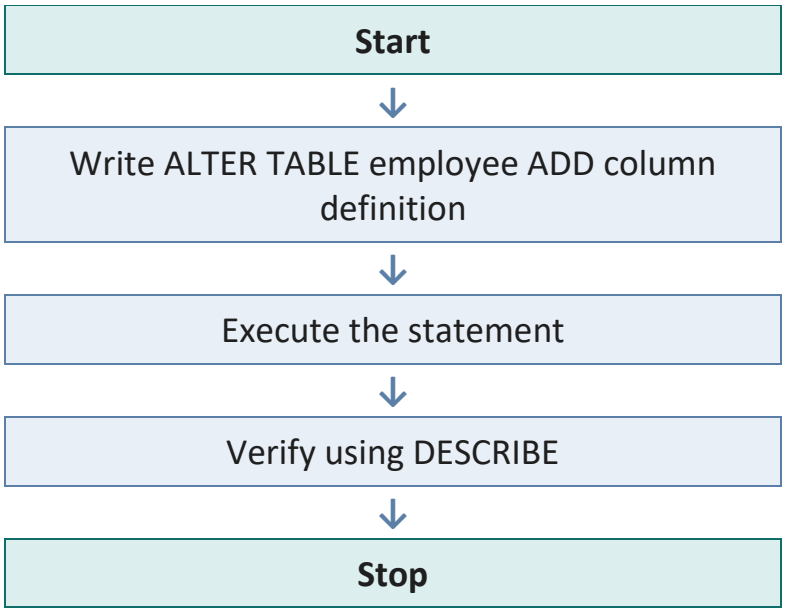
```
Query OK, 1 row affected
(Employee with EmpId 105 no longer appears in the
result set of SELECT *)
```

Query 47: ALTER TABLE - Add a Column

Aim:

To add a new column to an existing table structure using ALTER TABLE.

Flow Chart:



SQL Statement(s):

```
ALTER TABLE employee ADD Bonus DECIMAL(8,2) DEFAULT 0;
DESCRIBE employee;
```

Output:

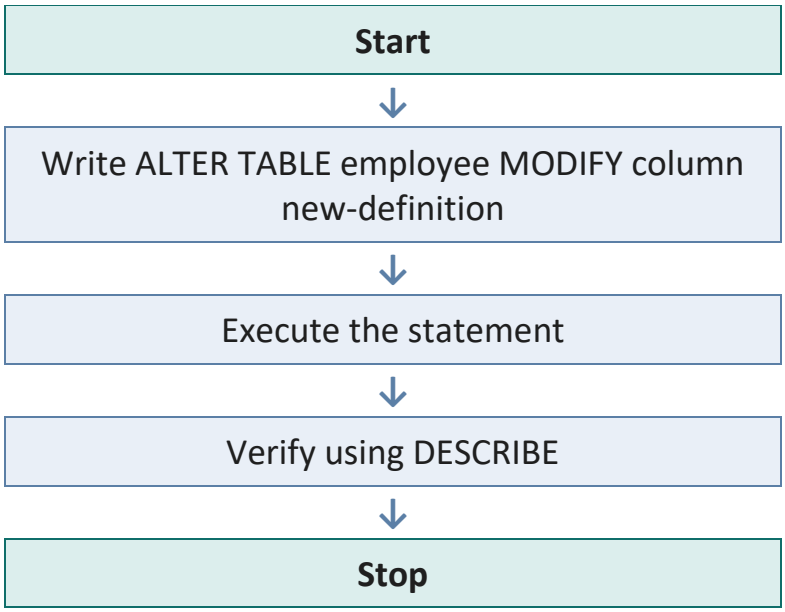
```
Query OK, 0 rows affected
(Table structure now shows an additional 'Bonus'
column of type DECIMAL(8,2))
```

Query 48: ALTER TABLE - Modify a Column

Aim:

To change the data type/size of an existing column using ALTER TABLE MODIFY.

Flow Chart:



SQL Statement(s):

```
ALTER TABLE employee MODIFY Ename VARCHAR(50);
DESCRIBE employee;
```

Output:

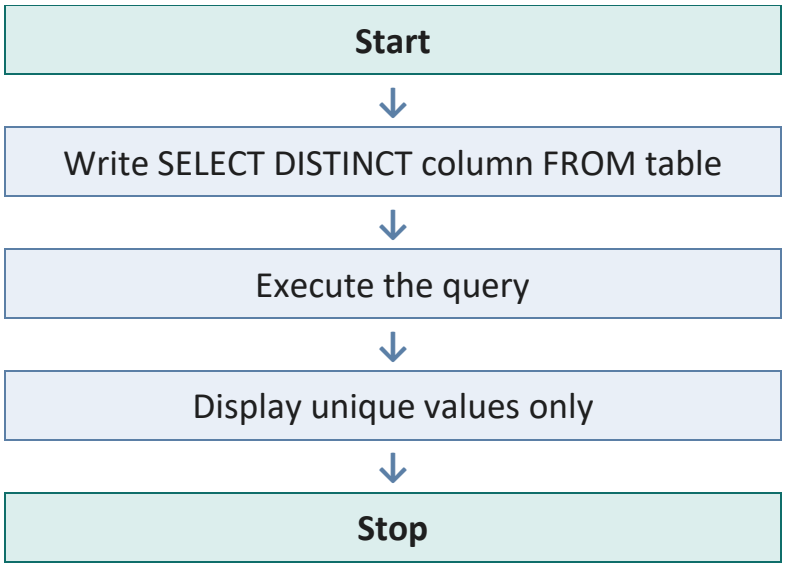
```
Query OK, 0 rows affected
(Column 'Ename' now shows type VARCHAR(50) in the
table structure)
```

Query 49: Use of DISTINCT

Aim:

To display unique values from a column, eliminating duplicate entries.

Flow Chart:



SQL Statement(s):

```
SELECT DISTINCT Dept FROM employee;
```

Output:

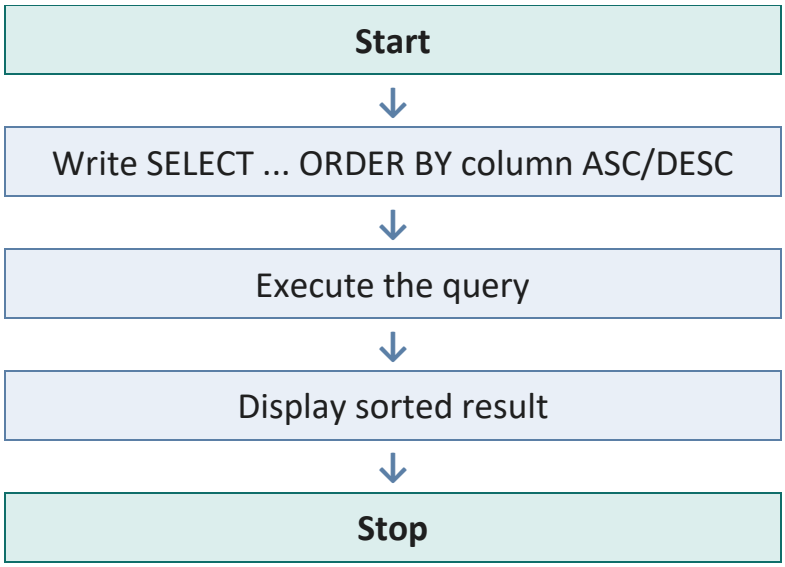
```
Dept
Sales
HR
IT
```

Query 50: ORDER BY (Ascending / Descending)

Aim:

To arrange query results in ascending or descending order using ORDER BY.

Flow Chart:



SQL Statement(s):

```
SELECT Ename, Salary FROM employee ORDER BY Salary DESC;
```

Output:

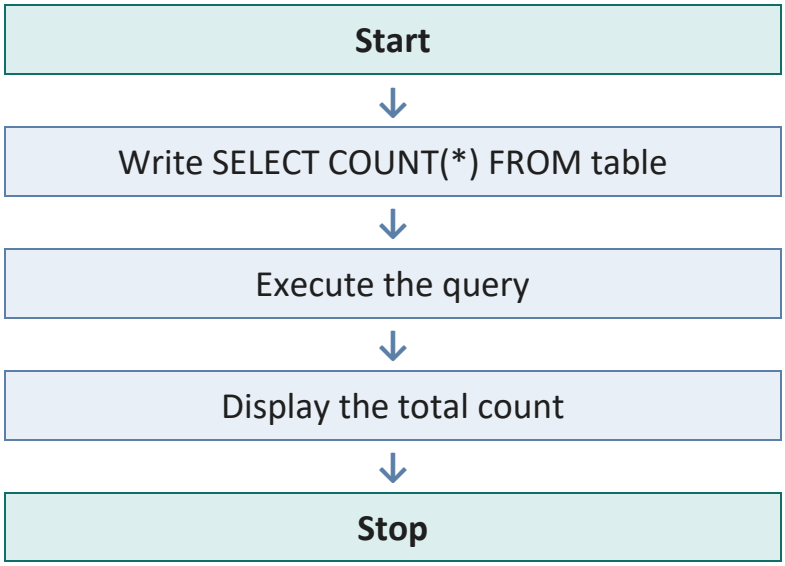
```
Ename         | Salary
Simran Kaur   | 77000
Karan Mehta   | 70000
Riya Kapoor   | 52000
Aman Sharma   | 45000
```

Query 51: Aggregate Function - COUNT()

Aim:

To count the number of records/rows using the COUNT() function.

Flow Chart:



SQL Statement(s):

```
SELECT COUNT(*) AS TotalEmployees FROM employee;
SELECT COUNT(DISTINCT Dept) AS TotalDepts FROM employee;
```

Output:

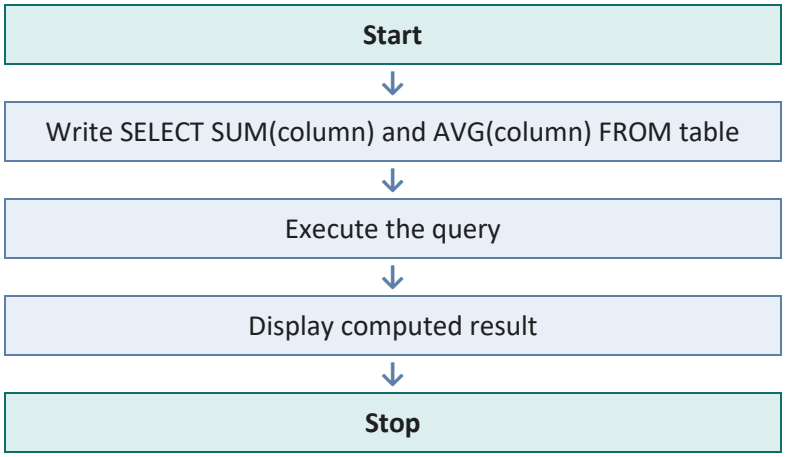
```
TotalEmployees
4
TotalDepts
3
```

Query 52: Aggregate Functions - SUM() and AVG()

Aim:

To compute the total and average value of a numeric column using SUM() and AVG().

Flow Chart:



SQL Statement(s):

```
SELECT SUM(Salary) AS TotalSalary, AVG(Salary) AS AvgSalary
FROM employee;
```

Output:

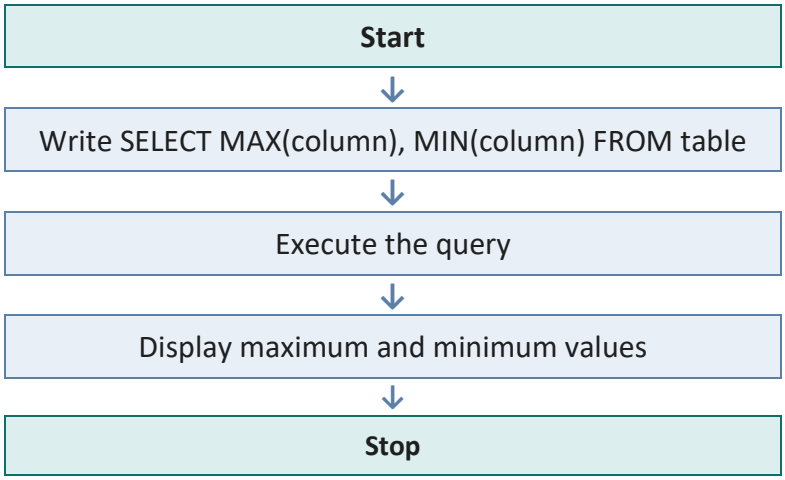
```
TotalSalary | AvgSalary
244000      | 61000.0000
```

Query 53: Aggregate Functions - MAX() and MIN()

Aim:

To find the highest and lowest values of a numeric column using MAX() and MIN().

Flow Chart:



SQL Statement(s):

```
SELECT MAX(Salary) AS Highest, MIN(Salary) AS Lowest
FROM employee;
```

Output:

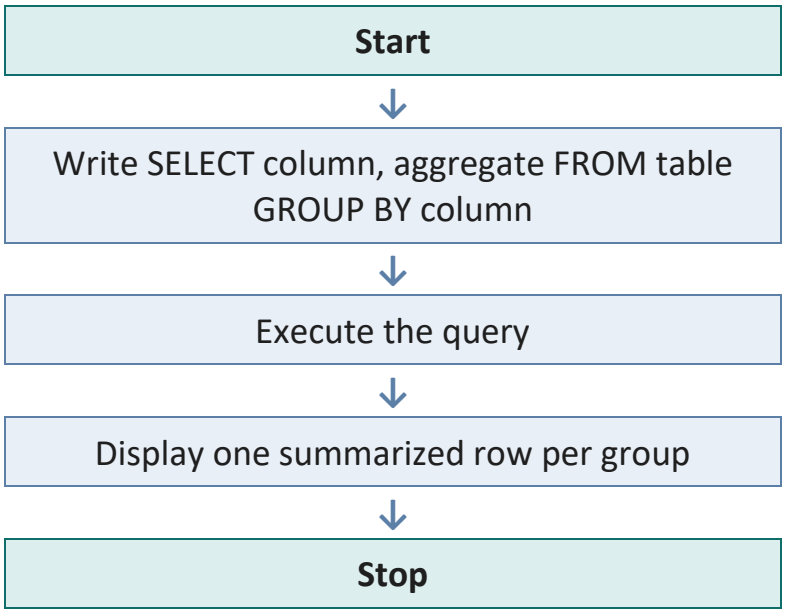
```
Highest | Lowest
77000   | 45000
```

Query 54: GROUP BY Clause

Aim:

To group rows sharing a common value and apply an aggregate function on each group.

Flow Chart:



SQL Statement(s):

```
SELECT Dept, COUNT(*) AS Employees, AVG(Salary) AS AvgSalary
FROM employee
GROUP BY Dept;
```

Output:

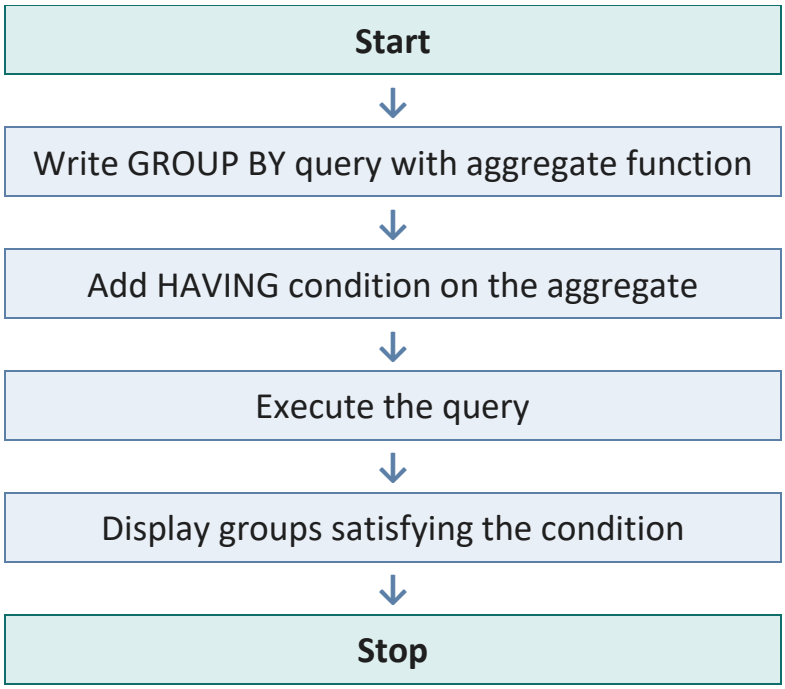
```
Dept  | Employees | AvgSalary
Sales | 1         | 45000.0000
HR    | 1         | 52000.0000
IT    | 2         | 73500.0000
```

Query 55: GROUP BY with HAVING Clause

Aim:

To filter grouped results using the HAVING clause (condition on aggregate value).

Flow Chart:



SQL Statement(s):

```
SELECT Dept, AVG(Salary) AS AvgSalary
FROM employee
GROUP BY Dept
HAVING AVG(Salary) > 50000;
```

Output:

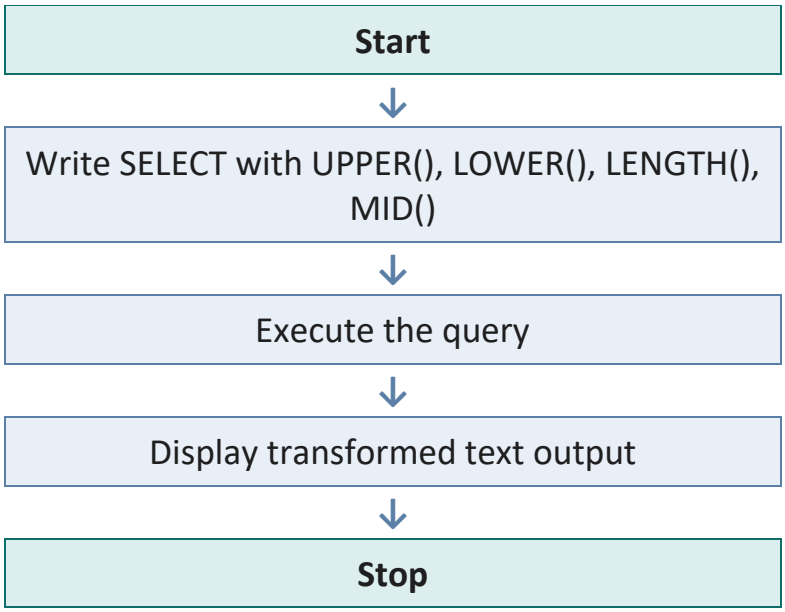
```
Dept | AvgSalary
HR   | 52000.0000
IT   | 73500.0000
```

Query 56: String Functions - UPPER, LOWER, LENGTH, SUBSTRING/MID

Aim:

To manipulate and extract text data using MySQL string functions.

Flow Chart:



SQL Statement(s):

```
SELECT Ename,
       UPPER(Ename) AS UpperName,
       LOWER(Dept) AS LowerDept,
       LENGTH(Ename) AS NameLength,
       MID(Ename, 1, 4) AS FirstFour
FROM employee;
```

Output:

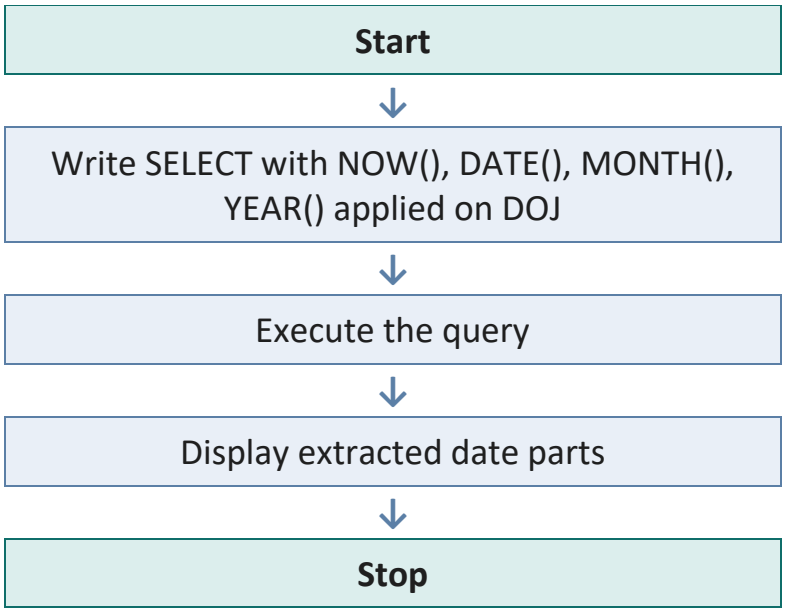
```
Ename         | UpperName    | LowerDept | NameLength |
FirstFour
Aman Sharma   | AMAN SHARMA  | sales     | 11         | Aman
Riya Kapoor   | RIYA KAPOOR  | hr        | 11         | Riya
Karan Mehta   | KARAN MEHTA  | it        | 11         | Kara
Simran Kaur   | SIMRAN KAUR  | it        | 11         | Simr
```

Query 57: Date Functions - NOW(), DATE(), MONTH(), YEAR()

Aim:

To extract and display date-related information using MySQL date functions.

Flow Chart:



SQL Statement(s):

```
SELECT Ename, DOJ, YEAR(DOJ) AS JoinYear,
       MONTH(DOJ) AS JoinMonth, NOW() AS CurrentDateTime
FROM employee;
```

Output:

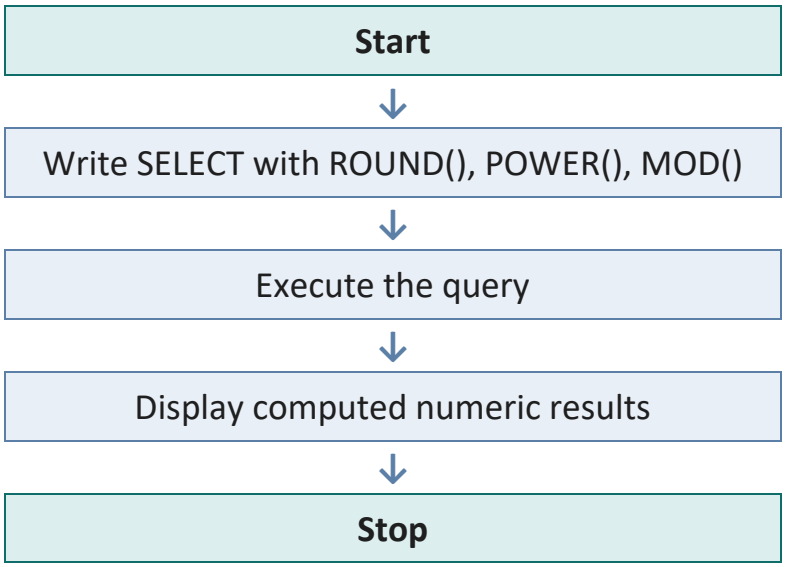
```
Ename        | DOJ        | JoinYear | JoinMonth |
CurrentDateTime
Aman Sharma  | 2019-06-12 | 2019     | 6         | 2026-08-11
10:30:00
Riya Kapoor  | 2020-03-05 | 2020     | 3         | 2026-08-11
10:30:00
Karan Mehta  | 2018-11-20 | 2018     | 11        | 2026-08-11
10:30:00
Simran Kaur  | 2021-01-15 | 2021     | 1         | 2026-08-11
10:30:00
```

Query 58: Math Functions - ROUND(), POWER(), MOD()

Aim:

To perform mathematical computations on numeric data using MySQL math functions.

Flow Chart:



SQL Statement(s):

```
SELECT Ename,
       ROUND(Salary/12, 2) AS MonthlySalary,
       POWER(2,3) AS PowerDemo,
       MOD(EmpId,10) AS LastDigit
FROM employee;
```

Output:

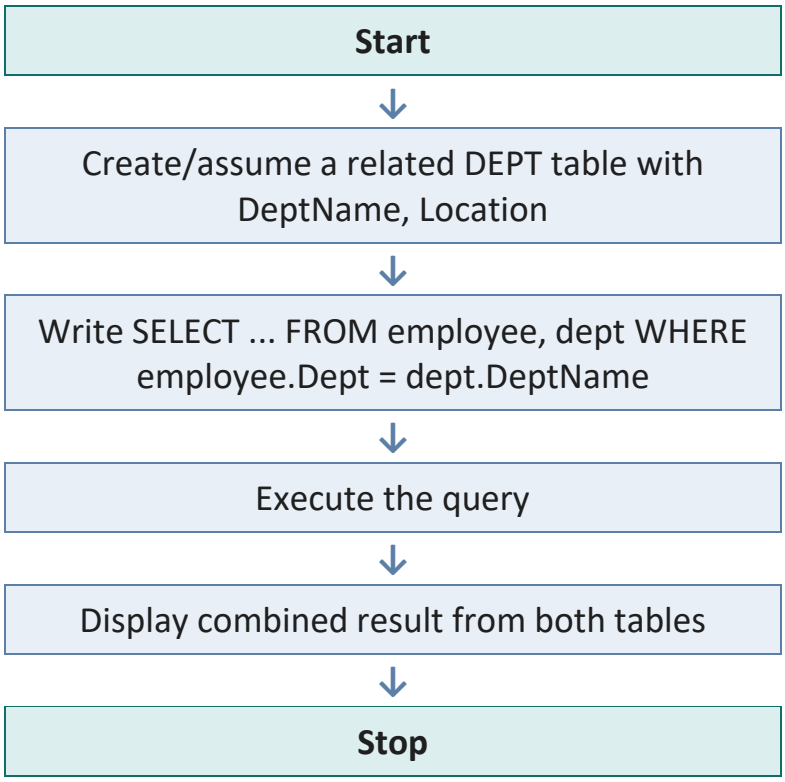
```
Ename         | MonthlySalary | PowerDemo | LastDigit
Aman Sharma   | 3750.00       | 8         | 1
Riya Kapoor   | 4333.33       | 8         | 2
Karan Mehta   | 5833.33       | 8         | 3
Simran Kaur   | 6416.67       | 8         | 4
```

Query 59: Equi-Join of Two Tables

Aim:

To combine records from two related tables (employee and dept) using an equi-join on a common column.

Flow Chart:



SQL Statement(s):

```
CREATE TABLE dept(
   DeptName VARCHAR(20) PRIMARY KEY,
   Location VARCHAR(30)
);
INSERT INTO dept
VALUES('Sales','Delhi'),('HR','Mumbai'),('IT','Bengaluru');

SELECT e.Ename, e.Dept, d.Location
FROM employee e, dept d
WHERE e.Dept = d.DeptName;
```

Output:

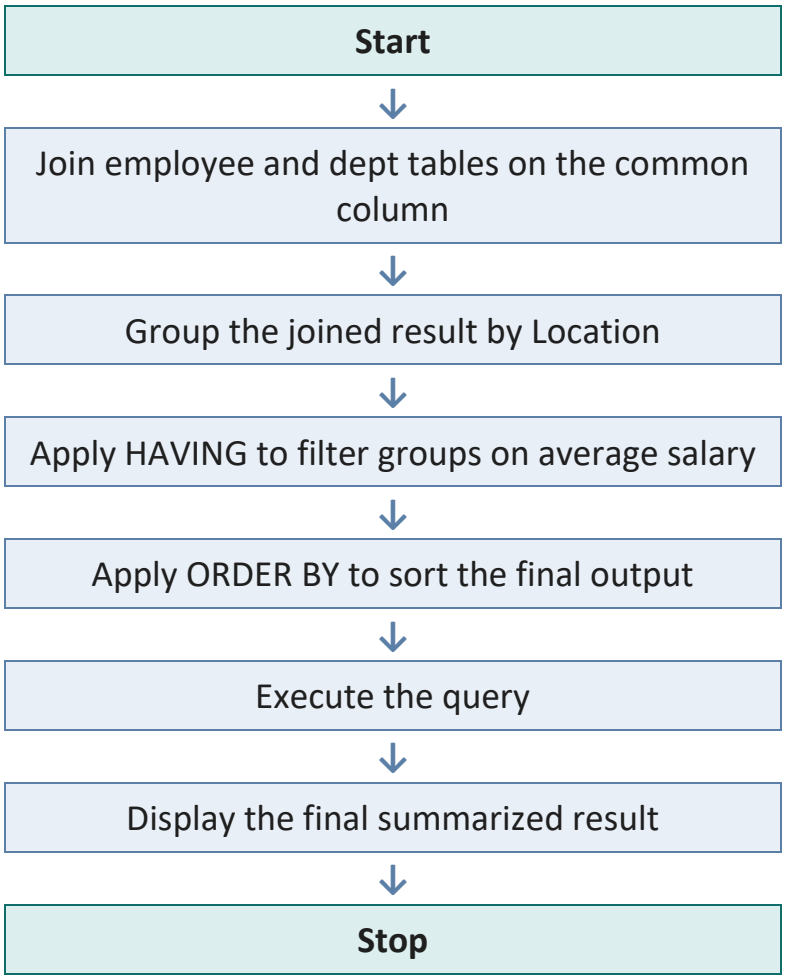
```
Ename         | Dept  | Location
Aman Sharma   | Sales | Delhi
Riya Kapoor   | HR    | Mumbai
Karan Mehta   | IT    | Bengaluru
Simran Kaur   | IT    | Bengaluru
```

Query 60: Complex Query: JOIN + GROUP BY + HAVING + ORDER BY

Aim:

To combine an equi-join with grouping, filtering and sorting in a single comprehensive query.

Flow Chart:



SQL Statement(s):

```
SELECT d.Location, COUNT(*) AS EmpCount, AVG(e.Salary) AS
AvgSalary
FROM employee e, dept d
WHERE e.Dept = d.DeptName
GROUP BY d.Location
HAVING AVG(e.Salary) > 50000
ORDER BY AvgSalary DESC;
```

Output:

```
Location    | EmpCount | AvgSalary
Bengaluru   | 2        | 73500.0000
Mumbai      | 1        | 52000.0000
```

PART D — Programming Questions in the Pattern of Previous Years' CBSE Board Examinations

10 Questions modelled on the format of recent CBSE Informatics Practices (065) board papers and sample papers: output-prediction, error-correction, SQL query writing, and case-study based questions

Q1. [Predict the Output (Series)]

Question:

```
Find and write the output of the following code:

import pandas as pd
s = pd.Series([5, 10, 15, 20], index=['p','q','r','s'])
print(s['q':'s'])
print(s*2)
```

Solution:

```
Output:
q    10
r    15
s    20
dtype: int64
p    10
q    20
r    30
s    40
dtype: int64

Explanation: Label-based slicing s['q':'s'] is INCLUSIVE of
the
end label, so it returns rows q, r and s. s*2 multiplies
every
element of the Series by 2 (vectorised operation).
```

Q2. [Find and Correct the Errors (DataFrame)]

Question:

```
Rewrite the following code after removing all syntax/logical
errors.
Underline each correction.

import pandas
df = pandas.Dataframe({'Name':['A','B'], 'Marks':[80,90])
print(df.Head())
```

Solution:

```
Corrected code:
import pandas as pd
df = pd.DataFrame({'Name':['A','B'], 'Marks':[80,90]})
print(df.head())

Corrections made:
1. 'Dataframe' -> 'DataFrame' (correct case of class name).
2. Missing closing '}' for the dictionary before ')' —
bracket mismatch corrected.
3. 'df.Head()' -> 'df.head()' (method names are case-
sensitive/lowercase).
```

Q3. [DataFrame Output-Based Question]

Question:

```
Consider the DataFrame df created as:

import pandas as pd
df = pd.DataFrame({'City':['Delhi','Pune','Agra','Delhi'],
                   'Sales':[200,150,120,300]})
print(df.groupby('City')['Sales'].sum())
print(df[df['Sales']>150])
```

Solution:

```
Output:
City
Agra     120
Delhi    500
Pune     150
Name: Sales, dtype: int64

    City  Sales
0  Delhi    200
3  Delhi    300

Explanation: groupby('City') combines the two 'Delhi' rows
and
sums their Sales (200+300=500). The filter
df[df['Sales']>150]
keeps only rows where Sales is strictly greater than 150.
```

Q4. [Write SQL Queries on a Given Table]

Question:

```
A table BOOKS(BookId, Title, Author, Price, Qty) is given.
Write SQL
queries for the following:
(a) Display all books with Price greater than 300, in
descending
    order of Price.
(b) Display the total number of books (SUM of Qty) author-
wise.
(c) Increase the Price of all books by 10%.
(d) Display Author names (without duplicates) who have
written
    more than one book.
```

Solution:

```
(a) SELECT * FROM BOOKS WHERE Price > 300 ORDER BY Price
DESC;

(b) SELECT Author, SUM(Qty) AS TotalQty FROM BOOKS GROUP BY
Author;

(c) UPDATE BOOKS SET Price = Price + (Price * 0.10);

(d) SELECT Author FROM BOOKS GROUP BY Author HAVING COUNT(*)
> 1;
```

Q5. [MySQL - Python Connectivity Output-Based Question]

Question:

```
A table 'employee' in MySQL database 'office' contains 5
records. A
Python program connects to it as follows. What will
cursor.rowcount
display, and what does the code accomplish?

import mysql.connector as sql
con = sql.connect(host="localhost", user="root",
password="root", database="office")
cursor = con.cursor()
cursor.execute("SELECT * FROM employee WHERE Salary > 40000")
data = cursor.fetchall()
print(len(data))
```

Solution:

```
The code connects Python to the MySQL database 'office',
executes
a SELECT query that filters employee records with Salary
greater
than 40000, and fetches all matching rows into the list
'data'
using fetchall(). print(len(data)) displays the COUNT of rows
returned by the query (i.e., the number of employees whose
salary
exceeds 40000) – not the total number of rows in the table.
```

Q6. [Predict the Output (Aggregate Functions in SQL)]

Question:

```
Given a table ORDERS(OrderId, Cust, Amount) with 6 rows where
three orders belong to customer 'Ravi' with amounts 500, 700,
300
and three belong to 'Meena' with amounts 900, 200, 400, what
will
the following query display?

SELECT Cust, COUNT(*), MAX(Amount), MIN(Amount)
FROM ORDERS
GROUP BY Cust
HAVING COUNT(*) >= 3;
```

Solution:

```
Output:
Cust  | COUNT(*) | MAX(Amount) | MIN(Amount)
Ravi  | 3        | 700         | 300
Meena | 3        | 900         | 200

Explanation: GROUP BY groups rows by customer; HAVING
COUNT(*)>=3
keeps only groups with 3 or more orders – here both customers
qualify since each has exactly 3 orders.
```

Q7. [Case-Study Based Question (DataFrame + Visualization)]

Question:

```
A retail store maintains a CSV file 'inventory.csv' with
columns
Item, Category, Stock. Write Python statements to:
(a) Read the CSV file into a DataFrame.
(b) Display items where Stock is less than 10 (low stock
alert).
(c) Plot a bar chart of Item vs Stock.
```

Solution:

```
import pandas as pd
import matplotlib.pyplot as plt

# (a)
df = pd.read_csv('inventory.csv')

# (b)
print(df[df['Stock'] < 10])

# (c)
plt.bar(df['Item'], df['Stock'], color='teal')
plt.title("Item-wise Stock")
plt.xlabel("Item")
plt.ylabel("Stock")
plt.show()
```

Q8. [Assertion-Reason / Conceptual Question]

Question:

```
State whether the following statement is True or False, with
a
reason: "The loc[] indexer in Pandas can only accept integer
position based indexes, similar to iloc[]."
```

Solution:

```
False.
Reason: loc[] is a LABEL-based indexer – it selects
rows/columns
using their index labels (which may be strings, dates, or
integers used as labels). iloc[] is the POSITION-based
indexer
that always uses integer positions (0,1,2,...) regardless of
the
actual index labels. They are complementary, not identical.
```

Q9. [Write a Python-MySQL Program (Full Program Question)]

Question:

```
Write a Python program to connect to a MySQL database
'college'
and delete the record of a student whose roll number is
entered
by the user, from the table 'student(rollno, name, marks)'.
```

Solution:

```
import mysql.connector as sql

con = sql.connect(host="localhost", user="root",
                  password="root", database="college")
cursor = con.cursor()

r = int(input("Enter Roll Number to delete: "))
cursor.execute("DELETE FROM student WHERE rollno = %s", (r,))
con.commit()

if cursor.rowcount > 0:
    print("Record deleted successfully")
else:
    print("Record not found")

con.close()
```

Q10. [Equi-Join Based Board Question]

Question:

```
Two tables are given:
STUDENT(RollNo, Name, StreamId)
STREAM(StreamId, StreamName)
Write an SQL query to display Name, StreamName for all
students,
using an equi-join.
```

Solution:

```
SELECT s.Name, st.StreamName
FROM STUDENT s, STREAM st
WHERE s.StreamId = st.StreamId;

-- Equivalent using explicit JOIN keyword:
SELECT s.Name, st.StreamName
FROM STUDENT s JOIN STREAM st
ON s.StreamId = st.StreamId;
```

Viva-Voce — Frequently Asked Questions

Quick revision questions commonly asked during the practical examination

Q1. What is the difference between a Pandas Series and a DataFrame?
Ans. A Series is a one-dimensional labelled array that can hold a single column of data, while a DataFrame is a two-dimensional labelled structure made up of multiple Series (columns), similar to a table.

Q2. What is the difference between loc[] and iloc[]?
Ans. loc[] selects data using index LABELS (inclusive of end label in slicing), while iloc[] selects data using integer POSITIONS (exclusive of end position), regardless of the actual labels.

Q3. Why is Matplotlib used in Informatics Practices?
Ans. Matplotlib is a Python library used to visualize data (from lists, Series or DataFrames) as line charts, bar charts, histograms, pie charts, scatter plots and box plots, making patterns and trends easier to interpret.

Q4. What is the purpose of the cursor object in Python-MySQL connectivity?
Ans. The cursor object is used to execute SQL statements and to fetch/traverse the result set returned by a query, after a connection to the database has been established.

Q5. Why is con.commit() necessary after INSERT/UPDATE/DELETE?
Ans. commit() permanently saves (persists) the changes made to the database. Without commit(), changes made via INSERT, UPDATE or DELETE may be rolled back or lost when the connection closes.

Q6. What is the difference between GROUP BY and HAVING?
Ans. GROUP BY groups rows that share a common value into summary rows; HAVING is used to filter these GROUPS based on a condition applied to an aggregate function, whereas WHERE filters individual rows before grouping.

Q7. What is an equi-join?
Ans. An equi-join combines rows from two tables based on a condition where values in a common column are EQUAL, using the '=' operator, e.g., table1.col = table2.col.

Q8. What does dropna() and fillna() do?
Ans. dropna() removes rows/columns containing missing (NaN) values, while fillna(value) replaces missing values with a specified value.

Q9. What is the difference between merge() and concat() in Pandas?
Ans. merge() combines DataFrames based on common column(s)/keys (like a SQL join), while concat() simply stacks DataFrames on top of each other (row-wise) or side-by-side (column-wise) without matching keys.

Q10. Why do we use executemany() instead of execute() in a loop?
Ans. executemany() allows multiple records to be inserted/updated in a single, efficient call by passing a list of tuples, instead of calling execute() repeatedly for each record.