CUET-UG Information Practices Sample Paper - 15

Duration: 1 Hour **Maximum Marks: 250**

Instructions

- This paper contains a total of 50 Multiple Choice Questions.
- Each correct answer carries **+5 marks**.
- Each incorrect answer carries **-1 mark**.
- No negative marking for unattempted questions.

Q1. Given a Pandas Series `S` with values [10, 20, 30, 40, 50] and indices ['a','b','c','d','e']. What will be the output of `print(S[1:4:2])`?

(A) b 20 , d 40

(B) a 10 , c 30

(C) b 20 , c 30

(D) a 20 , b 40

Q2. Consider a DataFrame `df` with 5 rows and 3 columns. If a user executes `df.iloc[1:3, 0:2] = 100`, how many elements in the DataFrame will be updated?

(A) 6

(B) 4

(C) 2

(D) 9

Q3. Which of the following code snippets will correctly display the last 3 rows of a DataFrame `df` having 100 rows?

(A) `df.head(-3)`

(B) `df.tail(3)`

(C) `df.iloc[97:100, :]`

(D) Both (B) and (C)

Q4. While importing data from a CSV file using `read_csv()`, which parameter is used to specify a custom column name if the file does not have a header row?

(A) `header=None, names=['Col1', 'Col2']`

(B) `colnames=['Col1', 'Col2']`

(C) `index_col=['Col1', 'Col2']`

(D) `usecols=['Col1', 'Col2']`

Q5. Identify the correct attribute to check the total number of elements in a Series `obj`.

(A) `obj.count()`

(B) `obj.size`

(C) `obj.shape`

(D) `obj.length`

Q6. A DataFrame `Result` has columns 'Subject' and 'Marks'. Which command will add a new column 'Grade' with a default value 'A' for all rows?

(A) `Result.add('Grade') = 'A'`

(B) `Result['Grade'] = 'A'`

(C) `Result.loc['Grade'] = 'A'`

(D) `Result.Grade('A')`

Q7. What will be the output of the following Pandas code?

```
import pandas as pd
s = pd.Series([1, 2], index=['x', 'y'])
print(s * 2 + s)
```

(A) x 3, y 6

(B) x 2, y 4

(C) x 4, y 8

(D) x 3, y 6 (as a Series)

Q8. To delete a column named 'Salary' from a DataFrame `df` permanently, which of the following is correct?

(A) `df.drop('Salary', axis=1)`

(B) `del df['Salary']`

(C) `df.remove('Salary')`

(D) `df.drop('Salary', axis=0, inplace=True)`

Q9. If a Series `S` contains [10, NaN, 30, NaN], what value is returned by `S.count()`?

(A) 4

(B) 2

(C) 0

(D) Error

Q10. What is the difference between `loc` and `iloc` in Pandas?

(A) `loc` is label-based; `iloc` is integer-position based.

(B) `loc` is integer-based; `iloc` is label-based.

(C) `loc` includes the start index but excludes the end; `iloc` includes both.

(D) There is no difference.

Q11. To export a DataFrame `df` to a file named `data.csv` without the row indices, which command is used?

(A) `df.to_csv('data.csv', index=False)`

(B) `df.to_csv('data.csv', header=False)`

(C) `df.save_csv('data.csv', index=None)`

(D) `df.write_csv('data.csv')`

Q12. Given a Series `Ser = pd.Series([5, 10, 15])`. What will be the result of `Ser > 7`?

(A) $[10, 15]$

(B) $[False, True, True]$

(C) $[True, True, True]$

(D) Error

Q13. In a DataFrame, `df.info()` provides:

(A) Statistical summary (mean, std, etc.)

(B) Data types and memory usage

(C) Only the first 5 rows

(D) Number of non-null values only

Q14. Which method is used to fill missing values (NaN) in a Pandas object with a specific value?

(A) `fillna()`

(B) `dropna()`

(C) `replace_nan()`

(D) `nullfill()`

Q15. How can you rename the index labels of a DataFrame `df`?

(A) `df.rename(index={old:new})`

(B) `df.relabel(index={old:new})`

(C) `df.index.name = 'new'`

(D) `df.columns = {old:new}`

Q16. Which command will sort the DataFrame `df` based on the column 'Age' in descending order?

(A) `df.sort_values('Age')`

(B) `df.sort('Age', ascending=False)`

(C) `df.sort_values('Age', ascending=False)`

(D) `df.arrange('Age', reverse=True)`

Q17. A Series `P` has indices $[0, 1, 2, 3]$. What is the result of `P.reindex([3, 2, 1, 0])`?

(A) It reverses the data values.

(B) It changes the index labels but keeps values in original positions.

(C) It rearranges the values to match the new order of labels.

(D) It results in an error.

Q18. In SQL, which clause is used to filter the results of an aggregate function like `SUM()` or `COUNT()`?

(A) `WHERE`

(B) `HAVING`

(C) `GROUP BY`

(D) `LIKE`

Q19. Consider a table `SALES` with columns `Region` and `Amount`. To find the total sales per region where the total is greater than 5000, the query is:

(A) `SELECT Region, SUM(Amount) FROM SALES WHERE SUM(Amount) > 5000 GROUP BY Region;`

(B) `SELECT Region, SUM(Amount) FROM SALES GROUP BY Region HAVING SUM(Amount) > 5000;`

(C) `SELECT Region, SUM(Amount) FROM SALES GROUP BY Amount HAVING Region > 5000;`

(D) `SELECT Region FROM SALES HAVING SUM(Amount) > 5000;`

Q20. Which type of JOIN returns all records from the left table and the matched records from the right table, filling with NULLs where there is no match?

(A) `INNER JOIN`

(B) `RIGHT JOIN`

(C) `LEFT JOIN`

(D) `FULL JOIN`

Q21. What is the correct order of clauses in a `SELECT` statement?

(A) `SELECT, FROM, HAVING, GROUP BY, WHERE`

(B) `SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY`

(C) `SELECT, WHERE, FROM, GROUP BY, ORDER BY`

(D) `SELECT, FROM, ORDER BY, WHERE, GROUP BY`

Q22. To sort the result of a query based on 'Salary' in descending order and then by 'Name' in ascending order, the clause is:

(A) `ORDER BY Salary DESC, Name`

(B) `ORDER BY Salary, Name DESC`

(C) `SORT BY Salary DESC, Name ASC`

(D) `ORDER BY Salary ASC, Name DESC`

Q23. How many rows will be returned by a Cartesian Product (Cross Join) of Table A (5 rows) and Table B (10 rows)?

(A) 15

(B) 5

(C) 50

(D) 10

Q24. Which keyword is used to eliminate duplicate rows from the result of a `SELECT` statement?

(A) `UNIQUE`

(B) `DISTINCT`

(C) `SINGLE`

(D) `CHECK`

Q25. When joining two tables `Emp` and `Dept` on `DeptID`, if the column `DeptID` exists in both tables, how do you refer to it in the `SELECT` clause?

(A) `DeptID`

(B) `Emp.DeptID`

(C) `DeptID.Emp`

(D) `Emp->DeptID`

Q26. The `COUNT(*)` function in SQL:

(A) Counts only non-null values in all columns.

(B) Counts all rows including those with NULL values.

(C) Counts only unique values in the first column.

(D) Counts only rows that have at least one NULL value.

Q27. What is the output of the SQL query: `SELECT ROUND(15.768, 2);`?

(A) 15.7

(B) 15.77

(C) 15.8

(D) 16.0

Q28. Which SQL function is used to return the current system date and time?

(A) `CURDATE()`

(B) `NOW()`

(C) `SYSDATE()`

(D) Both (B) and (C)

Q29. What will `SELECT INSTR('INFORMATICS', 'MA');` return?

(A) 5

(B) 6

(C) 4

(D) 0

(E) *Note: In SQL, indexing usually starts at 1.*

Q30. Which function is used to remove leading and trailing spaces from a string ' IP '?

(A) `CLEAN()`

(B) `TRIM()`

(C) `STRIP()`

(D) `LTRIM()`

Q31. To extract the name of the month from the date '2024-08-15', which function is used?

(A) `MONTHNAME()`

(B) `MONTH()`

(C) `EXTRACT_MONTH()`

(D) `DATE_MONTH()`

Q32. What will be the result of `SELECT POWER(3, 2) + MOD(10, 3);`?

(A) 10

(B) 11

(C) 9

(D) 1

Q33. The function `SUBSTR('ComputerScience', 9, 7)` will return:

(A) `Science`

(B) `Compute`

(C) `rScienc`

(D) `ScienceS`

Q34. Which function converts a string to all lowercase letters in MySQL?

(A) `LOWER()`

(B) `LCASE()`

(C) `SMALL()`

(D) Both (A) and (B)

Q35. In Matplotlib, which function is used to create a histogram?

(A) `plt.histogram()`

(B) `plt.barh()`

(C) `plt.hist()`

(D) `plt.plot(kind='hist')`

Q36. To add a label to the horizontal axis of a plot, we use:

(A) `plt.labelX()`

(B) `plt.xlabel()`

(C) `plt.axis(x='label')`

(D) `plt.set_x()`

Q37. Which parameter in `plt.pie()` is used to highlight or 'pop out' a specific slice of the pie?

(A) `highlight`

(B) `explode`

(C) `shadow`

(D) `autopct`

Q38. To display a legend on a graph, which function must be called after specifying 'label' in the plot functions?

(A) `plt.show_legend()`

(B) `plt.legend()`

(C) `plt.display()`

(D) `plt.label()`

Q39. What is the purpose of `plt.savefig('mygraph.png')`?

(A) To show the graph on the screen.

(B) To save the plot as an image file.

(C) To print the coordinates of the graph.

(D) To clear the current figure.

Q40. In Relational Algebra, the _____ operation is used to select a subset of columns from a table.

(A) Selection (σ)

(B) Projection (π)

(C) Union ($\cup$)

(D) Join ($\bowtie$)

Q41. A _____ is a minimal set of attributes that uniquely identifies a tuple in a relation.

(A) Foreign Key

(B) Candidate Key

(C) Secondary Key

(D) Composite Key

Q42. Which of the following is NOT a property of a Primary Key?

(A) It must be unique.

(B) It can contain NULL values.

(C) There can only be one primary key in a table.

(D) It uniquely identifies each record.

Q43. The _____ operation in relational algebra returns rows that are common to both relations.

(A) Set Difference

(B) Intersection

(C) Cartesian Product

(D) Selection

Q44. Which network topology requires a central controller or Hub to connect all nodes?

(A) Bus Topology

(B) Star Topology

(C) Ring Topology

(D) Mesh Topology

Q45. Identify the device that operates at the Data Link Layer and uses MAC addresses to forward data to specific ports.

(A) Hub

(B) Switch

(C) Repeater

(D) Router

Q46. What is the primary difference between a MAC address and an IP address?

(A) MAC is logical; IP is physical.

(B) MAC is 48-bit; IP (v4) is 32-bit.

(C) MAC addresses change with location; IP addresses are permanent.

(D) MAC is used for routing across the internet; IP is for local delivery.

Q47. Which term refers to the trail of data you leave behind while using the internet?

(A) Digital Footprint

(B) Digital Shadow

(C) Cyber Trail

(D) Data Cookies

Q48. Stealing someone else's intellectual work and representing it as your own without credit is called:

(A) Phishing

(B) Plagiarism

(C) Hacking

(D) Spamming

(E) *Note: This is related to IPR.*

Q49. The process of converting discarded electronic devices into reusable materials is known as:

(A) E-waste management

(B) Refurbishing

(C) Recycling

(D) Both (A) and (C)

Q50. Which law in India provides the legal framework for dealing with cybercrime and electronic commerce?

(A) Indian Penal Code

(B) IT Act 2000

(C) Consumer Protection Act

(D) Right to Information Act

Detailed Solutions

Q1.

Solution

Concept:

This question focuses on the fundamental concept of **Slicing in Pandas Series**. A Series is a one-dimensional labeled array. Slicing allows users to extract a specific subset of data based on index positions. The standard Python slicing syntax used in Pandas is `Series[start : stop : step]`.

1. **Start Index:** The position where the slice begins (inclusive).
2. **Stop Index:** The position where the slice ends (exclusive).
3. **Step:** The increment between each index in the slice.

In this specific problem, we are dealing with positional indexing (implicit) because the slice uses integers, even though the Series has explicit labels ('a', 'b', etc.).

Solution:

1. **Initialization:** We have a Series `S` with values $[10, 20, 30, 40, 50]$ mapped to indices $[a, b, c, d, e]$.

2. **Analyzing the Slice `[1:4:2]`:**

- **Start = 1**: This refers to the second element in the Series (Python uses 0-based indexing). The value at index 1 is `20` (associated with label 'b').
- **Stop = 4**: The slice will look up to, but not include, the element at position 4. Position 4 is the fifth element (value `50`, label 'e').
- **Step = 2**: Instead of moving one by one, the slice will skip one element.

3. **Execution**:

- Start at index `1`: Value is `20`, Label is `b`.
- Jump by 2 (next index is 1 + 2 = 3): Value at index `3` is `40`, Label is `d`.
- Next jump would be 3 + 2 = 5, which is beyond the stop index of `4`.

4. **Result Formation**: The resulting object is a new Series containing the rows for 'b' and 'd'.

Final Answer: The output will display the labels and values: `b 20` and `d 40`.

Answer: (A)

Q2.

Solution

Concept:

This question tests knowledge of **DataFrame attribute-based indexing and broadcasting** using the `.iloc` indexer. The `.iloc` indexer is purely integer-location based for selection by position. It follows the format `df.iloc[row_selection, column_selection]`.

When an assignment operator (`=`) is used with a slice of a DataFrame, Pandas performs an operation called "broadcasting." This means the scalar value provided (in this case, `100`) is assigned to every single cell identified within the specified range. Understanding the boundaries of the slice (start inclusive, stop exclusive) is critical for determining the number of elements affected.

Solution:

1. **Row Selection** `1:3`:

- In Python/Pandas slicing, the start index is included, and the stop index is excluded.
- Therefore, `1:3` selects row index `1` and row index `2`.
- Total rows selected = 2.

2. **Column Selection** `0:2`:

- Similarly, `0:2` selects column index `0` and column index `1`.
- It excludes column index `2`.
- Total columns selected = 2.

3. **Calculating Total Elements**:

- The intersection of the selected rows and columns forms a sub-grid or "block" within the DataFrame.
- Total elements = (Number of Rows) $\times$ (Number of Columns).
- Total elements = $2 \times 2 = 4$.

4. **Conclusion**:

- Even though the DataFrame has 5 rows and 3 columns (15 total elements), only the 4 cells located at (row 1, col 0), (row 1, col 1), (row 2, col 0), and (row 2, col 1) will be updated to the value `100`.

Final Answer: 4 elements will be updated.

Answer: (B)

Q3.

Solution

Concept:

This question explores the different methods available in Pandas to **retrieve the end portion of a dataset**. Accessing the last few rows is a frequent task during exploratory data analysis (EDA) to check for data entry consistency or the structure of the most recent records.

1. `tail(n)` **Method**: This is the most direct method. It returns the last `n` rows of the DataFrame.

2. `iloc` **Slicing**: Since `iloc` uses integer positions, it can be used to slice the DataFrame from a specific starting row index to the end.

3. `head(n)` **vs Negative indexing**: While `head(n)` returns the first `n` rows, `head(-n)` actually returns all rows *except* the last `n` rows, which is the opposite of what the question asks for.

Solution:

1. **Option (B) Analysis**: `df.tail(3)` is the built-in function designed specifically for this purpose. It will return the last 3 rows (indices 97, 98, and 99 in a 100-row DataFrame).

2. **Option (C) Analysis**: `df.iloc[97:100, :]` uses integer-based indexing.

- The starting point is index `97`.
- The stopping point is `100` (exclusive), so it stops at index `99`.
- Indices 97, 98, and 99 represent exactly 3 rows.
- The `:` for columns indicates that all columns should be included.
- Thus, (C) is also a valid way to retrieve the last 3 rows.

3. **Option (A) Analysis**: `df.head(-3)` would return the first 97 rows, which is incorrect.

4. **Conclusion**: Both `tail(3)` and the specific `iloc` slice result in the same output. In an exam context, recognizing that there are multiple ways to perform the same action in Pandas is key.

Final Answer: Both (B) and (C) are correct.

Answer: (D)

Q4.

Solution

Concept:

The `read_csv()` function is one of the most powerful and widely used functions in the Pandas library for **Data Ingestion**. By default, `read_csv()` assumes that the first row of the CSV file contains the column headers (names). However, real-world data files often come without headers or with headers that need to be overridden.

Two parameters are critical when handling files without headers:

1. `header`: This tells Pandas how to treat the first row. Setting `header=None` prevents Pandas from using the first data row as column names.

2. `names`: This allows the user to provide a list of strings to be used as column names. If `header=None` is set but `names` is not provided, Pandas assigns default integer names (0, 1, 2...).

Solution:

1. **Parameter `header=None`**: When we specify `header=None`, we are explicitly telling the parser that the file has no header row. If we omit this and only provide names, Pandas might mistakenly treat the first row of actual data as a header and then rename it, leading to data loss.

2. **Parameter `names`**: This parameter takes a list. For example, `names=['Col1', 'Col2']` will assign these labels to the columns.

3. **Evaluating other options**:

- `colnames` is not a valid parameter in `read_csv()` (though it exists in other languages like R).
- `index_col` is used to specify which column should be used as the row labels (index).
- `usecols` is used to filter which columns to read from the file, not to rename them.

4. **Correct Syntax**: The combination of `header=None` and `names=[...]` ensures the DataFrame is constructed correctly with the desired labels.

Final Answer: `header=None, names=['Col1', 'Col2']`

Answer: (A)

Q5.

Solution

Concept:

This question addresses **Series Attributes in Pandas**. Attributes are properties of an object that provide metadata about the data it contains. In the context of CUET-UG, it is vital to distinguish between "Methods" (which require parentheses, e.g., `.count()`) and "Attributes" (which do not, e.g., `.size`).

- **size**: Returns the total number of elements in the object.
- **count()**: Returns the number of non-null (non-NaN) elements.
- **shape**: Returns a tuple representing the dimensions.
- **ndim**: Returns the number of dimensions (always 1 for Series).

Solution:

1. **The 'size' Attribute**: For a Pandas Series, `size` returns an integer representing the total number of elements. It is equivalent to the length of the underlying data. Crucially, `size` includes `NaN` (Not a Number) or missing values.
2. **The 'count()' Method**: While `count()` also gives a number related to the size, it is a method that performs a calculation: it iterates through the data and only counts values that are not null. Therefore, if a Series has 10 elements and 2 are `NaN`, `size` will return 10, but `count()` will return 8.
3. **Attribute vs Method**: The question specifically asks for an "attribute" to check the "total number of elements." Since `size` is an attribute and provides the total count regardless of missing values, it is the correct choice.
4. **Shape and Length**: `shape` would return a tuple like `(5,)`, which describes the dimension rather than just the total number. `length` is not a valid attribute (though the Python built-in function `len(obj)` would work).

Final Answer: The correct attribute is `obj.size`.

Answer: (B)

Q6.

Solution

Concept:

This question focuses on **DataFrame Modification**, specifically the addition of a new column. In Pandas, DataFrames are size-mutable, meaning we can add or delete columns after the object is created. There are multiple ways to add a column, but the most common and intuitive method is the ”dictionary-style” assignment.

When you assign a scalar value (like a single character ’A’) to a new column name that does not yet exist in the DataFrame, Pandas performs **Broadcasting**. This means it automatically replicates that single value for every single row currently present in the DataFrame.

Solution:

1. **Dictionary-style Assignment**: By writing `Result['Grade'] = 'A'`, we are telling Pandas to create a new column labeled ’Grade’.

2. **Broadcasting Mechanism**: If the DataFrame `Result` has 50 rows, Pandas will not just put ’A’ in the first row; it will fill all 50 rows of the ’Grade’ column with ’A’.

3. **Evaluating Incorrect Options**:

- `Result.add('Grade')` is a method used for mathematical addition (like `df1.add(df2)`), not for schema modification.
- `Result.loc['Grade']` with a single label usually refers to a **row** index, not a column, unless specified in the second axis (`Result.loc[:, 'Grade']`).
- `Result.Grade('A')` looks like a method call, but ’Grade’ is not a built-in method of a DataFrame.

4. **Standard Practice**: The syntax `df['new_col'] = value` is the standard, most readable way to expand a DataFrame’s schema in Python for Data Science.

Final Answer: The correct command is `Result['Grade'] = 'A'`.

Answer: (B)

Q7.

Solution

Concept:

This question tests the understanding of **Vectorized Operations** in Pandas. One of the primary reasons for using Pandas (and NumPy) is the ability to perform operations on entire Series or DataFrames without writing explicit `for` loops.

When you perform arithmetic on a Series, Pandas applies that operation element-wise. This is highly efficient and follows the principles of linear algebra where a vector can be scaled or added to another vector.

Solution:

1. **Given Data**: We have a Series `s` where:

- Index 'x' has value 1.
- Index 'y' has value 2.

2. **The Expression `s * 2 + s`**:

- First, the multiplication `s * 2` is calculated. This results in a temporary Series where 'x' is 2 (1×2) and 'y' is 4 (2×2).
- Second, the original Series `s` is added to this temporary Series.
- For index 'x': 2(from $s * 2$) + 1(from s) = 3.
- For index 'y': 4(from $s * 2$) + 2(from s) = 6.

3. **Data Structure**: The result remains a Pandas Series object. It maintains the same alignment of indices ('x' and 'y').

4. **Conclusion**: The output will show the index and the calculated values. In an MCQ context, usually, the focus is on the numerical result, which is 3 and 6 respectively.

Final Answer: x 3, y 6 (as a Series).

Answer: (D)

Q8.

Solution

Concept:

This question deals with **Data Cleaning and Deletion** in DataFrames. Specifically, it distinguishes between "In-place" operations and operations that return a copy.

- **In-place**: Modifies the original object directly in memory.
- **Axis**: In Pandas, `axis=0` refers to rows, and `axis=1` refers to columns. To delete a column, we must specify the correct axis.
- **del Keyword**: This is a standard Python keyword used to remove items from a container (like a dictionary), and Pandas supports this for columns.

Solution:

1. **Option (B) Analysis**: `del df['Salary']` is a permanent, in-place operation. It directly removes the 'Salary' column from the DataFrame object.
2. **Option (A) Analysis**: `df.drop('Salary', axis=1)` will return a **new** DataFrame without the 'Salary' column, but the original `df` remains unchanged unless you assign it back (`df = df.drop(...)`) or use the `inplace=True` parameter.
3. **Option (D) Analysis**: This is incorrect because it specifies `axis=0`, which tells Pandas to look for a **row** named 'Salary', which does not exist in the context of columns.
4. **Option (C) Analysis**: There is no `.remove()` method for DataFrames; that is a method associated with Python Lists.
5. **Final Choice**: Since the question asks for a "permanent" deletion, `del` is the most straightforward direct answer among the options provided.

Final Answer: `del df['Salary']`

Answer: (B)

Q9.

Solution

Concept:

This question tests the distinction between **Total Elements and Non-Null Elements** in Pandas. Data in the real world is often "messy" and contains `NaN` (Not a Number) values representing missing data. Pandas provides specific methods to handle or identify these values.

The `count()` method is specifically designed to provide a count of "valid" data points. This is very different from the `size` attribute or the `len()` function, which count everything including the holes in the data.

Solution:

1. **Analyzing the Series**: The Series `S` contains:

- Index 0: 10 (Valid)
- Index 1: NaN (Null)
- Index 2: 30 (Valid)
- Index 3: NaN (Null)

2. **Applying `count()`**: When `S.count()` is executed, the internal logic of Pandas ignores any entries that are `np.nan` or `None`.

3. **Calculation**:

- Row 1: Yes
- Row 2: No
- Row 3: Yes
- Row 4: No
- Total valid count = 2.

4. **Comparison**: If the question had asked for `S.size`, the answer would have been 4. Because it asks for `.count()`, we only report the number of non-missing values. This distinction is a high-yield topic in CUET-UG exams.

Final Answer: 2

Answer: (B)

Q10.

Solution

Concept:

This is a fundamental concept in **Data Selection**. Pandas provides two primary ways to access data by position or label: `.loc` and `.iloc`. Understanding the difference is crucial for avoiding `KeyError` or `IndexError` during programming.

- `loc` is **Label-based**. You use the actual names of the rows and columns.
- `iloc` is **Integer-location based**. You use the numerical coordinates (starting from 0) regardless of what the labels are.

Solution:

1. **Primary Difference**: The core difference lies in how they interpret the inputs. If you have a row labeled 'Alpha', `df.loc['Alpha']` works. To get the same row with `iloc`, you would need to know its numerical position (e.g., `df.iloc[0]`).
2. **Slicing Behavior**: Another subtle but vital difference is that `loc` includes the last element in a slice (inclusive), whereas `iloc` follows standard Python slicing rules where the last element is excluded (exclusive).
3. **Use Case**: Use `loc` when you know the specific ID or Name of a record. Use `iloc` when you want to perform operations based on row numbers (e.g., "get the first 10 rows").
4. **Conclusion**: Option (A) correctly identifies that `loc` is for labels and `iloc` is for integer positions.

Final Answer: `loc` is label-based; `iloc` is integer-position based.

Answer: (A)

Q11.

Solution

Concept:

This question focuses on **Data Persistence**, specifically exporting a DataFrame to a CSV (Comma Separated Values) format. When working with Pandas, the `to_csv()` method is the standard tool for this task.

By default, Pandas includes the DataFrame's index (the row labels) as the first column in the output file. While this is often useful for internal data tracking, many external systems or machine learning models require a "clean" CSV containing only the data columns. To achieve this, we use the `index` parameter.

Solution:

1. **The `index` Parameter**: In the `to_csv()` function, setting `index=False` explicitly tells Pandas not to write row names (indices) into the CSV file.

2. **Evaluating Options**:

- `index=False` is the correct boolean flag for this purpose.
- `header=False` (Option B) would remove the column names (the first row), which is not what the question asked.
- `save_csv` and `write_csv` (Options C and D) are not valid Pandas methods.

3. **Execution Logic**: If your DataFrame has 3 columns (Name, Age, City) and 10 rows, using `index=False` results in a CSV with exactly 3 columns. Without it, you would get 4 columns, where the first column contains the numbers 0 through 9.

4. **Final Choice**: Option (A) provides the correct syntax for generating a clean data file.

Final Answer: `df.to_csv('data.csv', index=False)`

Answer: (A)

Q12.

Solution

Concept:

This question explores **Boolean Indexing and Masking**. Boolean indexing is a powerful feature in Pandas that allows you to filter data based on actual values rather than labels or positions. When a comparison operator (like >, <, or ==) is applied to a Series, Pandas performs an element-wise comparison. Instead of returning the values themselves, it returns a new Series of the same length containing Boolean values (`True` or `False`) based on whether the condition was met for each specific element.

Solution:

1. **Initialization**: `Ser = pd.Series([5, 10, 15])`.
2. **Step-by-Step Comparison**:

- Element 1 (5): Is 5 > 7? No. Result = `False`.
- Element 2 (10): Is 10 > 7? Yes. Result = `True`.
- Element 3 (15): Is 15 > 7? Yes. Result = `True`.

3. **Result Structure**: The output is a Series of Boolean values: `[False, True, True]`.
4. **Distinction**: It is important to note that `Ser[Ser > 7]` would return the actual values [10, 15], but the expression `Ser > 7` on its own returns the **mask** (the True/False values).

Final Answer: The result is `[False, True, True]`.

Answer: (B)

Q13.

Solution

Concept:

This question relates to **DataFrame Inspection**. Before performing analysis, a Data Scientist must understand the structure of the data. Pandas provides several methods for this, with `info()` and `describe()` being the most common.

While `describe()` focuses on statistical metrics (mean, min, max), `info()` focuses on the **technical metadata** of the DataFrame.

Solution:

1. **Contents of `info()`**: This method prints information about a DataFrame including:

- The index dtype and column dtypes.
- Non-null values (helping to identify missing data immediately).
- Memory usage (how much RAM the DataFrame is consuming).
- The total number of rows (RangeIndex) and total number of columns.

2. **Evaluating Options**:

- Option (A) refers to `describe()`.
- Option (C) refers to `head()`.
- Option (D) is only a partial truth, as `info()` provides much more than just the non-null count.

3. **Conclusion**: Option (B) correctly identifies the primary utility of `info()`, which is checking the data types (e.g., int64, object, float64) and the memory footprint.

Final Answer: Data types and memory usage.

Answer: (B)

Q14.

Solution

Concept:

Handling missing data is a critical step in the **Data Preprocessing** pipeline. In Pandas, missing data is usually represented as `NaN` (Not a Number). There are two main strategies: dropping the missing values entirely or filling them with a surrogate value (imputation).

The function used for imputation is `fillna()`. This method allows you to replace `NaN` values with a specific constant, the mean of the column, or even values from the previous/next row (forward/backward fill).

Solution:

1. **Methodology**: If you have a column 'Marks' with values $[80, NaN, 90]$, calling `df['Marks'].fillna(0)` would change the NaN to 0.

2. **Evaluating Options**:

- `fillna()` is the correct and official method name.
- `dropna()` is used to delete rows or columns containing missing values, not fill them.
- `replace_nan()` and `nullfill()` are not standard Pandas functions.

3. **In-place Parameter**: Like many Pandas methods, `fillna()` returns a copy by default. To modify the original DataFrame, one must use `inplace=True` or re-assign the result.

Final Answer: `fillna()`

Answer: (A)

Q15.

Solution

Concept:

This question focuses on **Relabeling and Renaming** in DataFrames. Often, column names or row indices provided in a raw dataset are not descriptive or contain errors. Pandas provides the `rename()` method to change specific labels without re-creating the entire index.

The `rename()` function is highly flexible because it uses a dictionary-like mapping: `{old_label : new_label}`.

Solution:

1. **Using `rename()`**: To change index labels, you specify the `index` parameter. For example, `df.rename(index={0: 'Row1'})` changes the label of the first row.

2. **Using `columns`**: Similarly, you can use `df.rename(columns={'A': 'Alpha'})` to rename columns.

3. **Evaluating Options**:

- Option (A) is the standard and correct way to map old labels to new ones.
- Option (C) `df.index.name` only changes the "title" of the index column (the header above the index), not the labels of the rows themselves.
- Option (B) `relabel()` is not a valid method name in Pandas.

4. **Alternative**: You can also rename all indices at once by assigning a list to `df.index`, but the `rename()` method is preferred when you only want to change a few specific values.

Final Answer: `df.rename(index={old:new})`

Answer: (A)

Q16.

Solution

Concept:

Sorting is a fundamental data manipulation task used to organize information in a meaningful sequence. In Pandas, the primary tool for this is the `sort_values()` method. Unlike basic Python lists which use `.sort()`, Pandas requires you to specify which column (or list of columns) should be used as the sorting key.

The `ascending` parameter is a boolean that controls the direction of the sort. By default, it is set to `True` (A-Z or 0-9). To sort in descending order (Z-A or 9-0), it must be explicitly set to `False`.

Solution:

1. **Method Identification**: The method `sort_values()` is the modern standard in Pandas. An older method `sort()` existed in very early versions but has since been deprecated and removed.
2. **Parameter Usage**: The first argument is typically the name of the column you wish to sort by.
3. **Logic Check**:
- `df.sort_values('Age')` would sort from youngest to oldest.
- `df.sort_values('Age', ascending=False)` sorts from oldest to youngest (descending).
4. **Evaluating Options**:
- (A) is incorrect because it defaults to ascending.
- (B) is incorrect because `sort()` is no longer a valid method.
- (D) is incorrect as `arrange()` is a function in the R language (dplyr), not Pandas.
5. **Result**: Therefore, Option (C) is the only syntactically correct and logical choice for the specified requirement.

Final Answer: `df.sort_values('Age', ascending=False)`

Answer: (C)

Q17.

Solution

Concept:

Reindexing is a core concept in Pandas that conforms a Series or DataFrame to a new set of labels. It is often misunderstood as ”renaming” or ”sorting,” but it is a distinct process of **Data Alignment**.

When you call `reindex()`, Pandas looks at the new labels provided. If a label in the new list matches an existing label, it moves the data associated with that label to the new position. If a new label is provided that wasn't there before, Pandas inserts `NaN`.

Solution:

1. **Initial State**: Series `P` has index $[0, 1, 2, 3]$ with corresponding values (let's assume V_0, V_1, V_2, V_3).

2. **Applying `reindex([3, 2, 1, 0])`:**

- The first new index is `3`. Pandas looks for index `3` in the original Series and finds V_3. It places V_3 first.
- The second new index is `2`. It finds V_2 and places it second.
- This continues until all labels in the new list are processed.

3. **Analyzing the Effect**: The values themselves are not ”reversed” by a math operation; they are rearranged so they stay attached to their original labels, but those labels are now in a different order.

4. **Conclusion**: Reindexing essentially rearranges the data to match the order of the newly provided labels. If the labels provided are a reverse of the original labels, the values will appear reversed, but logically, it is a ”matching and rearranging” operation.

Final Answer: It rearranges the values to match the new order of labels.

Answer: (C)

Q18.

Solution

Concept:

In SQL, filtering data can happen at two different stages of the query execution. Understanding the difference between `WHERE` and `HAVING` is one of the most common high-level assessment points in Informatics Practices.

- **WHERE**: Filters individual rows **before** any grouping or aggregation takes place.
- **HAVING**: Filters groups **after** the `GROUP BY` clause has been applied and aggregate functions (like `SUM`, `AVG`, `COUNT`) have been calculated.

Solution:

1. **Execution Order**: The SQL engine first identifies the table (`FROM`), then filters rows (`WHERE`), then groups the remaining rows (`GROUP BY`), then calculates the aggregates, and finally filters those results (`HAVING`).
2. **Constraint on WHERE**: You cannot use an aggregate function like `SUM(Salary)` inside a `WHERE` clause because at the `WHERE` stage, the computer hasn't calculated the sum yet; it is still looking at individual employees.
3. **The Role of HAVING**: If you want to see "Departments where the total salary is over 100,000", you must use `HAVING` because `100,000` is a property of the group (the department), not a property of a single row.
4. **Summary**: `HAVING` acts as a "WHERE clause for groups."

Final Answer: `HAVING`

Answer: (B)

Q19.

Solution

Concept:

This question requires the application of SQL clause logic to a specific business scenario: filtering aggregated data. When we need to find "total sales per region," we must use `GROUP BY Region`. When we add the condition "greater than 5000," we are filtering based on the result of an aggregate function (`SUM`).

As established in the previous question, conditions involving aggregate functions must be placed in the `HAVING` clause.

Solution:

1. **Requirement 1 (Total per Region)**: This necessitates the use of `SUM(Amount)` and `GROUP BY Region`.

2. **Requirement 2 (Filter condition)**: The condition `SUM(Amount) > 5000` is based on the aggregate.

3. **Evaluating the SQL syntax**:

- **Option A**: Uses `WHERE SUM(Amount) > 5000`. This will result in an "Invalid use of group function" error because aggregates are not allowed in `WHERE`.

- **Option B**: Uses `GROUP BY Region HAVING SUM(Amount) > 5000`. This is the correct syntax. It first groups the sales by region, calculates the sum for each, and then throws away regions that don't meet the 5000 threshold.

- **Option C**: Is logically incorrect because it groups by `Amount` (which would create a group for every unique price) and tries to filter `Region > 5000` (which makes no sense for text-based regions).

4. **Conclusion**: Option (B) follows the standard SQL "Big Six" order: Select-From-Where-Group-Having-Order.

Final Answer: `SELECT Region, SUM(Amount) FROM SALES GROUP BY Region HAVING SUM(Amount) > 5000;`

Answer: (B)

Q20.

Solution

Concept:

Joins are used to combine rows from two or more tables based on a related column between them. In Relational Databases, we often use different types of Joins to handle missing matches.

1. **Inner Join**: Returns only rows where there is a match in both tables.
2. **Left (Outer) Join**: Returns all rows from the ”left” table, even if there are no matches in the ”right” table.
3. **Right (Outer) Join**: Returns all rows from the ”right” table, even if there are no matches in the ”left” table.

Solution:

1. **The ”Left” Table**: In the syntax `TableA LEFT JOIN TableB`, TableA is the left table.
2. **Handling Mismatches**: If TableA has an employee in Department 99, but TableB (Departments) only goes up to Department 10, a `LEFT JOIN` will still show that employee.
3. **The Result of Mismatches**: Because there is no information in the right table for Department 99, the columns belonging to the right table will be filled with `NULL`.
4. **Scenario Matching**: The question describes a join that keeps all records from the left and only matches from the right, with NULLs for missing data. This is the textbook definition of a `LEFT JOIN` (or `LEFT OUTER JOIN`).
5. **Conclusion**: `LEFT JOIN` ensures that no data from the primary (left) source is lost, regardless of whether a corresponding entry exists in the secondary (right) source.

Final Answer: `LEFT JOIN`

Answer: (C)

Q21.

Solution

Concept:

The logical order of operations in an SQL `SELECT` statement is one of the most critical concepts in database management. While we write SQL in a specific way, the database engine executes the clauses in a distinct logical sequence to ensure data is filtered, grouped, and sorted efficiently. This sequence is often referred to as the "SQL Order of Execution."

The standard order for a full query is: `FROM` → `WHERE` → `GROUP BY` → `HAVING` → `SELECT` → `ORDER BY`. However, when writing the code (syntax), the structure must follow a specific rigid sequence.

Solution:

1. **The Starting Point**: Every retrieval query must start with `SELECT` (to define columns) followed immediately by `FROM` (to define the source table).
2. **Filtering Individual Rows**: The `WHERE` clause must come before any grouping or sorting. It reduces the dataset size early.
3. **Aggregation**: If the data needs to be categorized, `GROUP BY` follows `WHERE`. If those groups need filtering, `HAVING` follows `GROUP BY`.
4. **The Final Step**: `ORDER BY` is almost always the final clause in a standard query because you can only sort the data once it has been fetched, filtered, and aggregated.
5. **Evaluation of Option B**: This option correctly lists `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, and `ORDER BY` in the exact syntax order required by SQL engines like MySQL and PostgreSQL.

Final Answer: `SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY`

Answer: (B)

Q22.

Solution

Concept:

This question focuses on **Multi-level Sorting** using the `ORDER BY` clause. In real-world databases, it is common to have duplicate values in a primary sort column (e.g., several employees with the same salary). To resolve these "ties" and provide a predictable output, a secondary sort column is used.

The `ORDER BY` clause allows multiple column names separated by commas. Each column can have its own sort direction: `ASC` (Ascending, default) or `DESC` (Descending).

Solution:

1. **Primary Sort**: The requirement is to sort by 'Salary' in descending order. The syntax for this is `Salary DESC`.
2. **Secondary Sort**: For records where the Salary is identical, we must sort by 'Name' in ascending order. Since `ASC` is the default in SQL, we can simply write `Name` or `Name ASC`.
3. **Combining Clauses**: The correct way to write this is `ORDER BY Salary DESC, Name`.
4. **Evaluating Incorrect Options**:

- Option (B) sorts Salary ascending (default) and Name descending, which is the opposite of the requirement.
- Option (C) uses `SORT BY`, which is not the correct keyword in standard SQL (though used in HiveQL).
- Option (D) swaps the directions entirely.

5. **Logical Outcome**: If two people earn $50,000$, the one whose name starts with 'A' will appear before the one whose name starts with 'B' in this specific result set.

Final Answer: `ORDER BY Salary DESC, Name`

Answer: (A)

Q23.

Solution

Concept:

A **Cartesian Product** (also known as a `CROSS JOIN`) occurs when two tables are joined without any join condition or `WHERE` clause to link them. In this scenario, every single row from the first table is paired with every single row from the second table.

In relational algebra, if Table A has m rows and Table B has n rows, the Cartesian product $A \times B$ will result in a set of $(m \times n)$ rows. This concept is fundamental for understanding why missing join conditions in SQL queries lead to massive, often useless, result sets.

Solution:

1. **Given Data**:

- Rows in Table A = 5
- Rows in Table B = 10

2. **Calculation**:

- Row 1 of Table A pairs with all 10 rows of Table B $\rightarrow$ 10 rows.
- Row 2 of Table A pairs with all 10 rows of Table B $\rightarrow$ 10 rows.
- (Repeating for all 5 rows of A).
- Total Rows = $5 \times 10 = 50$.

3. **Columns**: It is also worth noting that the number of columns in the result will be the **sum** of the columns in both tables.

4. **Relevance**: In CUET-UG, identifying the cardinality of a Cartesian product is a common question to test the student's grasp of relational operations. If a table with 100 rows is cross-joined with a table of 1000 rows, the result is 100,000 rows, which can crash some applications.

Final Answer: 50

Answer: (C)

Q24.

Solution

Concept:

When querying a database, it is common to find redundant data across rows. For example, in a table of customers, the 'City' column might contain 'Delhi' hundreds of times. If a user wants to see a list of unique cities where customers reside, they must use a keyword to filter out the duplicates from the output.

The keyword `DISTINCT` is used in the `SELECT` clause to return only unique values for the specified columns.

Solution:

1. **The Role of** `DISTINCT`: When `SELECT DISTINCT City FROM Customers` is executed, the SQL engine scans the results and, if it finds a value it has already encountered, it discards the duplicate.

2. **Evaluating Options**:

- **DISTINCT**: The standard SQL keyword for unique results.
- **UNIQUE**: This is a constraint used when creating a table (to prevent duplicates from being entered), but it is not used in a `SELECT` statement to filter output.
- **CHECK**: A constraint used to limit the range of values in a column.

3. **Scope**: `DISTINCT` applies to all columns listed in the `SELECT` clause. If you select `DISTINCT City, State`, SQL only removes rows where *both* the City and State are identical to a previous row.

4. **Performance**: While useful, `DISTINCT` can be slower on very large datasets because it requires the database to sort or hash the data to find duplicates.

Final Answer: `DISTINCT`

Answer: (B)

Q25.

Solution

Concept:

This question deals with **Column Ambiguity** in SQL joins. When joining multiple tables, it is highly likely that different tables will share identical column names (e.g., `ID`, `Name`, or `DeptID`). If you try to select `DeptID` without specifying which table it belongs to, the SQL engine will return an "Ambiguous column name" error because it doesn't know whether you want the `DeptID` from the `Emp` table or the `Dept` table. To resolve this, we use **Dot Notation** (Table Qualification).

Solution:

1. **Dot Notation Syntax**: The format is `TableName.ColumnName`. This tells the engine exactly where to look.

2. **Application**: In the join of `Emp` and `Dept`, the column `DeptID` exists in both. Therefore, to refer to it in the `SELECT` or `ON` clause, you must write `Emp.DeptID` or `Dept.DeptID`.

3. **Table Aliases**: To make queries shorter, developers often use aliases: `SELECT e.DeptID FROM Emp e JOIN Dept d ON e.DeptID = d.DeptID`.

4. **Evaluating Options**:

- Option (B) `Emp.DeptID` is the standard, syntactically correct way to qualify a column.
- Option (D) `Emp->DeptID` is syntax used in some programming languages like C++ or PHP for pointers/objects, but never in standard SQL.

5. **Best Practice**: Even if a column name is unique, using table qualifications in complex joins is considered a best practice for readability and maintenance.

Final Answer: `Emp.DeptID`

Answer: (B)

Q26.

Solution

Concept:

This question explores the behavior of the `COUNT()` aggregate function, specifically the distinction between `COUNT(*)` and `COUNT(column_name)`. In database management, `NULL` values represent missing or unknown information. Different functions treat these `NULL`s differently.

- `COUNT(column_name)`: Counts only the non-null entries in that specific column.
- `COUNT(*)`: Counts the total number of rows in the table that satisfy the query criteria, regardless of whether individual cells contain data or are empty.

Solution:

1. **Row Counting**: When you use `COUNT(*)`, the SQL engine treats the entire row as a single unit. It does not check if specific attributes within that row are `NULL`.
2. **Logic**: If a table has 10 rows, and in one row every single column is `NULL`, `COUNT(*)` will still return 10. This is because the row (the tuple) exists in the database schema.
3. **Use Case**: This is the most efficient way to determine the size of a result set or the total number of records in a table.
4. **Comparison**: If we had a table of 'Students' where 5 students haven't provided their 'Email', `COUNT(Email)` would return 5 less than `COUNT(*)`.
5. **Conclusion**: Option (B) correctly identifies that `COUNT(*)` is inclusive of all rows, including those that might be entirely or partially comprised of `NULL` values.

Final Answer: Counts all rows including those with NULL values.

Answer: (B)

Q27.

Solution

Concept:

Mathematical functions in SQL, such as `ROUND()`, are essential for formatting numerical data for reports. The `ROUND()` function takes two arguments: the number to be rounded and the number of decimal places (precision) to round to.

The logic follows standard mathematical rounding rules: if the digit after the specified precision is 5 or greater, the preceding digit is rounded up. If it is less than 5, it remains the same.

Solution:

1. **The Input**: We are rounding the value `15.768` to 2 decimal places.
2. **The Process**:

- Look at the first two decimal places: `.76`.
- Look at the third decimal place (the digit immediately following our target precision): `8`.
- Since `8` is greater than or equal to `5`, we must round up the second decimal place.

3. **Calculation**: The `6` in the second decimal place becomes `7`.
4. **Result**: The final value is `15.77`.
5. **Edge Cases**: If the query was `ROUND(15.768, 0)`, the result would be `16`. If it was `ROUND(15.768, 1)`, the result would be `15.8`.

Final Answer: 15.77

Answer: (B)

Q28.

Solution

Concept:

Working with temporal data (dates and times) is a core requirement for database applications like banking or reservation systems. SQL provides several built-in functions to retrieve the current time from the server's operating system.

While different SQL dialects (MySQL, Oracle, SQL Server) have their own preferences, many functions are shared or serve identical purposes.

Solution:

1. `NOW()` **Function**: In MySQL, `NOW()` returns the current date and time in 'YYYY-MM-DD HH:MM:SS' format.
2. `SYSDATE()` **Function**: This function also returns the current date and time. The subtle difference is that `NOW()` returns the time at which the query *started* executing, while `SYSDATE()` returns the actual time at which the function *executes* (which can be different in long-running queries).
3. `CURDATE()` **Function**: This function returns only the current date ('YYYY-MM-DD') without the time component.
4. **CUET-UG Context**: For the purpose of Informatics Practices, both `NOW()` and `SYSDATE()` are recognized as functions that provide the current date and time.
5. **Final Choice**: Since both (B) and (C) satisfy the requirement of providing both date and time, Option (D) is the correct answer.

Final Answer: Both (B) and (C)

Answer: (D)

Q29.

Solution

Concept:

The `INSTR()` (In-String) function is a string manipulation tool used to find the starting position of a substring within a larger string. It is vital for text processing and data validation tasks.

A critical rule in SQL (specifically MySQL) is that **string indexing starts at 1**, unlike Python where indexing starts at 0. This is a common trap for students who are used to Pandas or general Python programming.

Solution:

1. **Function Call**: `INSTR('INFORMATICS', 'MA')`.

2. **Search Process**:

- Position 1: I
- Position 2: N
- Position 3: F
- Position 4: O
- Position 5: R
- Position 6: M
- Position 7: A

3. **Match Found**: The substring 'MA' begins at position 6 (where 'M' is located).

4. **Verification**:

I(1) N(2) F(3) O(4) R(5) **M(6)** A(7) T(8) I(9) C(10) S(11).

5. **Conclusion**: The function returns the integer `6`. If the substring was not found at all, the function would return `0`.

Final Answer: 6

Answer: (B)

Q30.

Solution

Concept:

Data entered by users often contains ”noise” in the form of accidental leading or trailing spaces (e.g., ” Admin ”). These spaces can cause issues during string comparisons (e.g., ”Admin” is not the same as ” Admin”).

To handle this, SQL provides ”Trim” functions.

- `LTRIM()`: Removes leading spaces (Left side).
- `RTRIM()`: Removes trailing spaces (Right side).
- `TRIM()`: Removes spaces from both sides.

Solution:

1. **Requirement**: The question asks to remove *both* leading and trailing spaces from the string ’ IP ’.

2. **Evaluating `TRIM()`**: This is the universal function for removing surrounding whitespace. Executing `SELECT TRIM(' IP ');` will result in the string ’IP’.

3. **Evaluating Options**:

- `CLEAN()` is not a standard SQL string function.
- `STRIP()` is the equivalent method in Python, but it is not used in SQL.
- `LTRIM()` would only remove the spaces on the left, leaving the trailing spaces intact (’IP ’).

4. **Usage**: `TRIM` is also capable of removing specific characters if configured (e.g., `TRIM(LEADING '0' FROM '000123')`), but its default behavior is to target spaces.

Final Answer: `TRIM()`

Answer: (B)

Q31.

Solution

Concept:

This question focuses on **Date Extraction functions** in SQL. Databases store dates in a standard internal format (usually 'YYYY-MM-DD'). To present this data in a human-readable or categorized format, we use specific functions that can isolate parts of the date.

- `MONTH()`: Returns the numerical month (1 for January, 12 for December).
- `MONTHNAME()`: Returns the full string name of the month (e.g., 'August').
- `DAYNAME()`: Returns the name of the day (e.g., 'Thursday').

Solution:

1. **Analysis of the Input**: The date provided is '2024-08-15'.
2. **Requirement**: The user wants the "name" of the month.
3. **Function Application**:

- `SELECT MONTHNAME('2024-08-15');`
- The middle part of the date is '08', which corresponds to August.
- The function will return 'August'.

4. **Evaluating Options**:

- Option (B) `MONTH()` would return the number `8`.
- Options (C) and (D) are not valid standard MySQL/SQL functions.

5. **Context**: This is extremely useful in business reports where "Sales in August" sounds more professional than "Sales in Month 8."

Final Answer: `MONTHNAME()`

Answer: (A)

Q32.

Solution

Concept:

This question tests the ability to combine multiple **Mathematical Functions** in a single SQL expression. It involves exponentiation (`POWER`) and the remainder operator (`MOD`).

- `POWER(base, exponent)`: Calculates $base^{exponent}$.
- `MOD(n, d)`: Calculates the remainder of n divided by d.

Solution:

1. **Evaluating** `POWER(3, 2)`:
- $3^2 = 3 \times 3 = 9$.
2. **Evaluating** `MOD(10, 3)`:
- $10 \div 3$ is 3 with a remainder of 1 (since $3 \times 3 = 9$ and $10 - 9 = 1$).
- The result is 1.
3. **Final Addition**:
- Result = 9(from Power) + 1(from Mod).
- Result = 10.
4. **Importance**: These functions are often used in Informatics Practices for data transformation, such as calculating interest rates or determining if a value is even/odd (using `MOD(x, 2)`).

Final Answer: 10

Answer: (A)

Q33.

Solution

Concept:

The `SUBSTR()` (or `SUBSTRING()`) function is used to extract a specific portion of a string. It requires three parameters: the source string, the starting position, and the number of characters to extract.

As a reminder, SQL uses **1-based indexing**. The syntax is `SUBSTR(string, start_position, length)`.

Solution:

1. **Input String**: 'ComputerScience'.
2. **Start Position**: 9.

- C(1) o(2) m(3) p(4) u(5) t(6) e(7) r(8) **S(9)**.
- The extraction starts at 'S'.

3. **Length**: 7 characters.

- Count 7 characters starting from 'S': S(1) c(2) i(3) e(4) n(5) c(6) e(7).

4. **Result**: The extracted string is 'Science'.
5. **Common Error**: Students often mistake the third parameter for the "end position" (like in Python). In SQL, it is the count of characters. If it were the end position, the result would be different.

Final Answer: `Science`

Answer: (A)

Q34.

Solution

Concept:

In SQL, casing functions are used to standardize text data. This is particularly important for search queries where 'Admin' and 'admin' might be treated differently by the database.

Most SQL dialects provide redundant functions to perform the same task to ensure compatibility across different systems.

Solution:

1. `LOWER()`: This is the standard ANSI SQL function to convert a string to lowercase.
2. `LCASE()`: This is a synonym specifically used in MySQL to perform the exact same task.
3. **Example**: `LOWER('HELLO')` and `LCASE('HELLO')` both result in 'hello'.
4. **Opposite Functions**: The uppercase equivalents are `UPPER()` and `UCASE()`.
5. **Conclusion**: Since both (A) and (B) are valid and commonly used in the IP syllabus, Option (D) is the correct choice.

Final Answer: Both (A) and (B)

Answer: (D)

Q35.

Solution

Concept:

This question shifts to **Data Visualization using Matplotlib**. A histogram is a specific type of plot used to represent the distribution of a continuous variable. It groups data into ”bins” and shows the frequency of data points within each bin.

While the `plot()` function is a general-purpose command, Matplotlib provides specialized functions for specific chart types to ensure better control over the parameters.

Solution:

1. **The `plt.hist()` function**: This is the primary command for creating histograms. It automatically calculates the frequency of the data and plots the bars.

2. **Evaluating Options**:

- `plt.histogram()` is a common mistake; the actual function name is the shortened `hist()`.
- `plt.barh()` is used to create horizontal bar charts, not histograms.
- `plt.plot(kind='hist')` is the syntax used in **Pandas** plotting, but the question asks about **Matplotlib** directly.

3. **Key Arguments**: `plt.hist()` often takes a second argument, `bins`, which determines how many intervals the data should be divided into.

Final Answer: `plt.hist()`

Answer: (C)

Q36.

Solution

Concept:

When creating visualizations in Matplotlib, providing context for the data is essential for clarity. Labels allow the viewer to understand what variables are being plotted on each axis.

In Matplotlib's `pyplot` module, specific functions are dedicated to setting these labels. For the horizontal axis, we refer to the "x-axis," and for the vertical axis, we refer to the "y-axis."

Solution:

1. **Function Identification**: The function `plt.xlabel()` is the standard command to set the label for the x-axis. It takes a string as an argument, which then appears centered below the horizontal axis of the plot.

2. **Evaluating Options**:

- `plt.labelX()` is not a valid Matplotlib function.
- `plt.ylabel()` would be the counterpart for the vertical axis.
- `plt.axis()` is used to set or get the limits of the axes (e.g., `plt.axis([xmin, xmax, ymin, ymax])`), not for labeling them.

3. **Implementation**: A typical usage would look like `plt.xlabel("Year")` or `plt.xlabel("Temperature in C")`.

4. **Formatting**: You can also pass additional arguments to `xlabel` to change the font size, color, or weight, such as `plt.xlabel("Sales", fontsize=12, color='red')`.

Final Answer: `plt.xlabel()`

Answer: (B)

Q37.

Solution

Concept:

A **Pie Chart** is used to represent the proportional size of items (categories) in a single data series. Sometimes, for emphasis, a user might want to pull one or more slices away from the center of the chart.

In Matplotlib, this effect is achieved using a specific parameter within the `plt.pie()` function.

Solution:

1. **The `explode` Parameter**: This parameter takes a list or an array of values, where each value represents the fraction of the radius by which to offset each wedge.

2. **Usage**: If you have a pie chart with 4 slices and you want to highlight the second one, you would pass `explode=[0, 0.1, 0, 0]`. The second slice will be "exploded" out by 10% of the radius.

3. **Evaluating Options**:

- `shadow` is a boolean parameter (`True/False`) that adds a 3D shadow effect to the pie but does not move the slices.
- `autopct` is used to format and display the percentage values inside the wedges.
- `highlight` is not a valid parameter for the `pie()` function.

4. **Visual Impact**: Using `explode` is a standard data storytelling technique to draw the viewer's eye to a specific category, such as the "Other" category or a leading competitor's market share.

Final Answer: `explode`

Answer: (B)

Q38.

Solution

Concept:

A **Legend** is a crucial component of a graph, especially when multiple data series are plotted on the same axes. It identifies what each color or line style represents.

However, simply providing a label in the plotting function (like `plt.plot(x, y, label='Sales')`) is not enough to make the legend appear. You must explicitly tell Matplotlib to draw the legend box.

Solution:

1. **Step 1: Labeling**: When calling a plot function (like `bar`, `hist`, or `plot`), you must include the `label` argument.
2. **Step 2: Activation**: After all plots are defined, you must call `plt.legend()`. This function looks for all labeled plots and creates a small box on the graph matching labels to colors/styles.
3. **Customization**: The `legend()` function can take arguments like `loc` (location) to place the box in the 'upper right', 'lower left', or 'center'.
4. **Evaluating Options**:

- `plt.show_legend()` and `plt.display()` are not the correct function names.
- `plt.show()` is used to render the final figure to the screen, but it won't add a legend if `plt.legend()` wasn't called first.

Final Answer: `plt.legend()`

Answer: (B)

Q39.

Solution

Concept:

Data analysis often involves sharing results with others. While `plt.show()` is useful for interactive work (like in a Jupyter Notebook), it does not save the result to a persistent file on the hard drive.

The `savefig()` function is the primary method used to export plots into various image formats such as PNG, PDF, SVG, or JPG.

Solution:

1. **Functionality**: `plt.savefig('filename.extension')` takes the current state of the plot and writes it to a file.

2. **Parameters**: You can control the resolution of the image using the `dpi` (dots per inch) parameter. For example, `plt.savefig('mygraph.png', dpi=300)` creates a high-quality image suitable for printing.

3. **Evaluation**:

- Option (A) refers to `plt.show()`.
- Option (D) refers to `plt.clf()` (clear figure).
- Option (B) correctly identifies that `savefig()` is for file creation.

4. **Important Note**: In many environments, `plt.savefig()` must be called **before** `plt.show()`, because `show()` may clear the current figure after it is closed.

Final Answer: To save the plot as an image file.

Answer: (B)

Q40.

Solution

Concept:

Relational Algebra is a theoretical language used to describe operations on relational databases. Two of the most fundamental operations are **Selection** and **Projection**.

- **Selection (σ):** Filters **rows** based on a condition (horizontal subset). - **Projection (π):** Selects specific **columns** from a table (vertical subset).

Solution:

1. **Understanding Projection**: If you have a table `Student` with columns (RollNo, Name, Age, Address), and you only want to see a list of names, you perform a "Projection" on the 'Name' column.

2. **Mathematical Notation**: It is represented by the Greek letter Pi (π). For example, $\pi_{Name}(Student)$.

3. **SQL Equivalent**: In SQL, the `SELECT` clause (without the WHERE clause) is the implementation of Projection. For example, `SELECT Name FROM Student`.

4. **Evaluating Options**:

- **Selection**: This would be used if you wanted "Students with Age > 15."
- **Union**: Used to combine rows from two compatible tables.
- **Join**: Used to combine columns from two different tables based on a key.

5. **Conclusion**: Since the question specifically asks about choosing a "subset of columns," Projection is the correct terminology.

Final Answer: Projection (π)

Answer: (B)

Q41.

Solution

Concept:

In a Relational Database Management System (RDBMS), **Keys** are used to establish and identify relationships between tables and to ensure that each record within a table is uniquely identifiable. There are several types of keys:

1. **Super Key**: A set of one or more attributes that can uniquely identify a tuple.
2. **Candidate Key**: A minimal Super Key; it has no redundant attributes.
3. **Primary Key**: One of the candidate keys selected by the database designer to be the unique identifier.
4. **Alternate Key**: Candidate keys that were not chosen as the Primary Key.

Solution:

1. **Defining Candidate Key**: A Candidate Key is a column, or set of columns, in a table that can uniquely identify any database record without referring to any other data.
2. **The "Minimal" Requirement**: The key word in the definition is "minimal." For example, if `Roll_No` is unique, then (`Roll_No`, `Name`) is a Super Key, but it is not a Candidate Key because `Name` is redundant—`Roll_No` alone is sufficient.
3. **Evaluating Options**:

- **Foreign Key**: Used to link two tables together; it doesn't have to be unique in its own table.
- **Secondary Key**: Usually refers to a non-primary key used for indexing purposes.
- **Composite Key**: A key that consists of more than one attribute. While a Candidate Key *can* be composite, it is not the definition of the unique identifier set.

4. **Conclusion**: The definition provided in the question matches the standard technical definition of a Candidate Key.

Final Answer: Candidate Key

Answer: (B)

Q42.

Solution

Concept:

A **Primary Key** is the most important constraint in a database table. It ensures **Entity Integrity**. For a column to be designated as a Primary Key, it must adhere to strict rules set by the relational model. These rules ensure that we can always find and identify a specific row without ambiguity.

Solution:

1. **Uniqueness**: Every value in the Primary Key column must be unique. No two rows can share the same ID.

2. **Non-Nullability**: This is the most crucial rule. A Primary Key **cannot contain NULL values**. If a Primary Key were NULL, we would have no way to reference or identify that specific row, which violates the fundamental principles of data integrity.

3. **Singularity**: A table can have only one Primary Key. While that key can consist of multiple columns (Composite Primary Key), there is only one "Primary Key" definition per table.

4. **Evaluating Options**:

- Option (A) and (D) are true properties of a Primary Key.
- Option (C) is a true rule of RDBMS.
- Option (B) is false. If you try to insert a NULL value into a Primary Key column in SQL, the database will return an error.

5. **Final Choice**: Since the question asks for the property that is NOT true, Option (B) is the correct answer.

Final Answer: It can contain NULL values.

Answer: (B)

Q43.

Solution

Concept:

Relational algebra includes "Set Operations" because tables are essentially sets of tuples. Common set operations include Union, Intersection, and Set Difference.

- **Union** ($\cup$): Combines all rows from both tables (removing duplicates).
- **Intersection** ($\cap$): Returns only the rows that appear in both Table A and Table B.
- **Difference** ($-$): Returns rows in Table A that are not in Table B.

Solution:

1. **Logic of Intersection**: Imagine a table of 'Students in Sports' and a table of 'Students in Music'. If you want to find students who are in both groups, you are looking for the intersection of those two sets.

2. **Union Compatibility**: For the Intersection operation to work, both tables must be "Union Compatible," meaning they must have the same number of columns and the corresponding columns must have compatible data types.

3. **Evaluating Options**:

- **Selection**: Filters rows based on a logic condition (e.g., Grade > 90).
- **Cartesian Product**: Combines every row of A with every row of B.
- **Intersection**: Correctly describes the process of finding common records.

4. **SQL Equivalent**: In SQL, this is performed using the `INTERSECT` keyword (available in most modern SQL engines).

Final Answer: Intersection

Answer: (B)

Q44.

Solution

Concept:

Network Topology refers to the physical or logical arrangement of nodes (computers, printers, etc.) in a network. Choosing the right topology affects the cost, complexity, and reliability of the network.

The **Star Topology** is currently the most popular arrangement used in Local Area Networks (LANs), especially those using Ethernet with a central switch or hub.

Solution:

1. **Structure of Star Topology**: Every node is connected to a central device (Hub, Switch, or Router) via a point-to-point connection. The nodes do not connect directly to each other.

2. **Role of the Hub/Controller**: All data traffic passes through the central controller. If Computer A wants to send data to Computer B, it sends it to the Hub, which then forwards it to Computer B.

3. **Evaluating Options**:

- **Bus Topology**: Uses a single common cable (the backbone) to which all nodes are connected. There is no central controller.
- **Ring Topology**: Each node is connected to exactly two other nodes, forming a continuous circle.
- **Mesh Topology**: Every node is connected to every other node (Full Mesh), or at least some nodes have multiple connections (Partial Mesh).

4. **Advantages of Star**: If one cable fails, only that node is disconnected. The rest of the network continues to function. However, if the central Hub fails, the entire network goes down.

Final Answer: Star Topology

Answer: (B)

Q45.

Solution

Concept:

Networking devices are specialized hardware used to manage data flow across a network. To understand them, we often refer to the **OSI Model**.

1. **Hub**: A physical layer device that broadcasts data to all ports (inefficient).
2. **Switch**: A data link layer device that is "smarter" than a hub.
3. **Router**: A network layer device that connects different networks and uses IP addresses.

Solution:

1. **The Switch**: A Switch maintains a **MAC Address Table**. When a data packet (frame) arrives, the switch looks at the destination MAC address.
2. **Intelligent Forwarding**: Unlike a hub, which sends the data to everyone, the switch looks at its table and sends the data only to the specific port where that MAC address is located.
3. **Evaluating Options**:

- **Repeater**: Simply regenerates a signal to extend its range; it does not look at addresses.
- **Router**: Uses IP addresses (Logical addresses) to move data between different networks (e.g., your home to the Internet).
- **Hub**: Uses no addresses; it just repeats bits out of every port.

4. **Conclusion**: The device that uses MAC addresses to forward data specifically to the destination port is the Switch.

Final Answer: Switch

Answer: (B)

Q46.

Solution

Concept:

Communication in a network relies on two types of addresses: the **Physical Address (MAC)** and the **Logical Address (IP)**.

1. **MAC Address**: A 48-bit unique identifier burned into the Network Interface Card (NIC) by the manufacturer. It is used for local delivery within a network segment.

2. **IP Address**: A 32-bit (IPv4) or 128-bit (IPv6) address assigned by the network software. It is used to identify a device across different networks (the Internet).

Solution:

1. **Size Difference**: A standard MAC address is 48 bits, usually represented in hexadecimal (e.g., `00:1A:2B:3C:4D:5E`). An IPv4 address is 32 bits, represented in dotted decimal (e.g., `192.168.1.1`).

2. **Nature**: MAC addresses are permanent to the hardware, while IP addresses change depending on which network the device is connected to (Dynamic vs Static).

3. **Evaluating Options**:

- Option (A) is false; it's the other way around.
- Option (C) is false; IPs change, MACs don't.
- Option (D) is false; IP is used for global routing.
- Option (B) correctly identifies the bit lengths of both address types.

4. **Importance**: In CUET-UG, understanding that MAC works at the Data Link Layer and IP works at the Network Layer is a key differentiator for high-scoring students.

Final Answer: MAC is 48-bit; IP (v4) is 32-bit.

Answer: (B)

Q47.

Solution

Concept:

A **Digital Footprint** is the unique set of traceable digital activities, actions, contributions, and communications manifested on the Internet or digital devices. Every time you post on social media, visit a website, or send an email, you leave a trail behind.

There are two types:

1. **Active**: Data you intentionally share (e.g., posting a photo).

2. **Passive**: Data collected without your active involvement (e.g., websites tracking your IP address or browsing history).

Solution:

1. **Definition**: The question describes the ”trail of data” left behind. This is the literal definition of a digital footprint.

2. **Evaluating Options**:

- **Digital Shadow**: Often used interchangeably with passive digital footprints, but ”Footprint” is the comprehensive term used in the NCERT/Informatics Practices syllabus.

- **Cyber Trail**: A generic term, but not the formal technical term used in academics.

- **Data Cookies**: These are small files stored on your computer that *contribute* to your footprint, but they are not the footprint itself.

3. **Consequences**: It is important to realize that digital footprints are often permanent. Once data is public (or stored on a server), the user has little control over how it is used by third parties, advertisers, or even future employers.

Final Answer: Digital Footprint

Answer: (A)

Q48.

Solution

Concept:

Intellectual Property Rights (IPR) protect the creations of the mind. When someone uses another person's ideas, writing, or artistic work without proper attribution and claims it as their own, they are committing an ethical and sometimes legal violation.

This specific violation is called **Plagiarism**. It is especially common in academic and digital environments where "copy-pasting" is technically easy.

Solution:

1. **Understanding Plagiarism**: Plagiarism is the "wrongful appropriation" and "stealing and publication" of another author's "language, thoughts, ideas, or expressions."

2. **Evaluating Options**:

- **Phishing**: A cybercrime where a target is contacted by email/phone by someone posing as a legitimate institution to lure individuals into providing sensitive data.
- **Hacking**: Gaining unauthorized access to data in a system or computer.
- **Spamming**: Sending the same message indiscriminately to a large number of recipients.

3. **Plagiarism vs. Copyright Infringement**: While related, plagiarism is a matter of ethics and academic integrity (claiming credit), whereas copyright infringement is a legal matter (violating the owner's exclusive rights).

4. **Prevention**: To avoid plagiarism, one must always use citations and give credit to the original source.

Final Answer: Plagiarism

Answer: (B)

Q49.

Solution

Concept:

E-waste (Electronic Waste) consists of discarded electrical or electronic devices. Used electronics which are destined for refurbishment, reuse, resale, salvage recycling through material recovery, or disposal are also considered e-waste.

The informal processing of e-waste in developing countries can lead to adverse human health effects and environmental pollution. Therefore, proper management is crucial.

Solution:

1. **Recycling**: This is the process of breaking down discarded devices into raw materials (like gold, copper, and plastic) which can then be used to manufacture new products.

2. **E-waste Management**: This is the broader term that encompasses the collection, transportation, and processing of electronic waste to minimize its environmental impact.

3. **Evaluating Options**:

- **Refurbishing**: This involves repairing and cleaning a device to be sold again. While helpful, it isn't the process of converting waste into "materials."

- **Recycling** and **E-waste Management** both describe the professional handling of these materials.

4. **Conclusion**: Since management includes the recycling process and both are focused on reusability and environmental safety, Option (D) is the most comprehensive answer.

Final Answer: Both (A) and (C)

Answer: (D)

Q50.

Solution

Concept:

As digital technology and the internet became central to commerce and communication, India needed a legal framework to govern digital activities and punish cybercrimes. This led to the enactment of the **Information Technology Act, 2000**.

The Act provides legal recognition for transactions carried out by means of electronic data interchange and other means of electronic communication.

Solution:

1. **The IT Act 2000**: This is the primary law in India dealing with cybercrime and electronic commerce. It was later amended in 2008 to include more specific crimes like ”cyber terrorism” and ”data protection.”

2. **Key Sections**:

- **Section 66A** (later struck down): Dealt with offensive messages.
- **Section 66C**: Identity theft.
- **Section 66D**: Cheating by personation by using computer resources.

3. **Evaluating Options**:

- **Indian Penal Code (IPC)**: While many cybercrimes are also prosecuted under the IPC, the IT Act is the specific legislation for electronic matters.
- **RTI Act**: Deals with the right to access information from public authorities.
- **Consumer Protection Act**: Deals with the rights of consumers in general trade.

4. **Relevance**: For an Informatics Practices student, knowing the IT Act 2000 is fundamental for understanding the ”Societal Impacts” unit of the syllabus.

Final Answer: IT Act 2000

Answer: (B)

Answer Key

Q	Ans	Q	Ans	Q	Ans	Q	Ans	Q	Ans
1	A	2	B	3	D	4	A	5	B
6	B	7	D	8	B	9	B	10	A
11	A	12	B	13	B	14	A	15	A
16	C	17	C	18	B	19	B	20	C
21	B	22	A	23	C	24	B	25	B
26	B	27	B	28	D	29	B	30	B
31	A	32	A	33	A	34	D	35	C
36	B	37	B	38	B	39	B	40	B
41	B	42	B	43	B	44	B	45	B
46	B	47	A	48	B	49	D	50	B