**First and last name** ..............................................................................................

## Question 1/22 *(2 p.)*

Which of the following statements are true regarding **List / Array** type columns in a SQL (Postgres) database? **Select all that apply.**

*A.* All elements of the array should be of the **same data type and multi-dimensional** arrays are allowed.
*B.* Given a list of elements, it is possible to check if they **exist** in the column using a **query**
*C.* Arrays in PostgreSQL must be of **fixed size**, and the size must be **declared** at the time of **table creation.**
*D.* PostgreSQL supports **Linear Algebra** operations out-of-the-box on an array column.

## Question 2/22 *(2 p.)*

Which of the following statements are **true** regarding **JSON** / **JSONB** type columns in a SQL (Postgres) database? **Select all that apply.**

*A.* It is **not possible** to use the standard **comparison** operators (**<, =, >, etc.**) with **JSONB** data.
*B.* It is possible to extract values using **in-built JSON functions** and **operators** from a **JSON/JSONB column** by querying its **keys**.
*C.* It is possible to construct complex & functional queries **to extract** information from **nested** or **hierarchical JSON/JSONB** data
*D.* Table constraints (such as **foreign keys** or **unique constraints) can** be applied **directly** to individual elements **(keys)** within a **JSON/JSONB** column.

## Question 3/22 *(2 p.)*

A **join** query is producing **duplicate rows**, and you are considering using the **DISTINCT** keyword to remove them. Under which circumstances is DISTINCT considered a bad practice, and what would be the correct solution? **Select all that apply.**

*A.* When the table **sizes** are **small**. Solution: use **groupby.**
*B.* When duplicate rows are due to a **bug in the join** condition; Solution: Correcting the join condition to fix the underlying issue.
*C.* When there are **duplicate records** in one of the **tables**. Solution: Set a primary **key constraint** on the **tables**, or **deduplicate** the results in a **subquery**
*D.* When duplicate rows are due to a **bug in the join** condition; Solution: Set a primary **key constraint** on the **tables**

## Question 4/22

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Transpose **matrix** using list comprehension. Which code snippet accomplishes this?

*A.* `[[row[i] for row in matrix] for i in range(len(matrix))]`

*B.* `[[row[i] for i in range(len(matrix[0]))] for row in matrix]`

*C.* `[[row[i] for row in matrix] for i in range(len(matrix[0]))]`

## Question 5/22 *(3 p.)*

Given below are two equivalent pieces of code in Pandas & Apache Spark. Use any one snippet to arrive at the answer.

**Pandas**

```
df['A'] = df['A'].explode()
df['A'] = df['A'] * 2
```

**Spark**

```
df = df.withColumn('A', F.explode('A'))
df = df.withColumn('A', df['A'] * 2)
```

Assume that column **A** contains a list of integers per row.

*A.* The code transforms column **A** by assigning each integer of the list to their **own separate columns** and multiplies these values by 2.

*B.* The code transforms column **A** by assigning each integer of the list to its **own separate row** in column A and multiplies these values by 2.

*C.* The code **doubles the length of all list values** in column **A**.

*D.* The code **broadcasts all list values** across the entire column **A** so that every row has a union of the entire column's list elements multiplied by 2.

## Question 6/22 *(3 p.)*

Given below are two equivalent pieces of code in Pandas & Apache Spark. Use any one snippet to arrive at the answer. **Select all options that apply.**

**Pandas**

```
df = pd.json_normalize(pd.DataFrame(data)['B'])
```

**Spark**

```
df = spark.read.json(spark.sparkContext.parallelize(data))
df = df.select(df["B.*"])
```

Assume that **data** is a variable as follows:

```
data = [
    {'A': 'foo', 'B': {'C': 3, 'D': 'bar'}},
    {'A': 'baz', 'B': {'C': 4, 'D': 'qux'}}
]
```

*A.* The code **normalizes** JSON data by converting it into **hexadecimal representation** for easier processing, storage and compression.

*B.* The df variable will have columns **A,B,C,D**

*C.* The code **flattens** a JSON document as a usable table

*D.* If the JSON is deeply nested (for ex. if ***D*** has a JSON instead of a string), then one **column** is created for **each depth variant** present in the input JSON data.

*E.* When we know the **structure** of the JSON data **in advance**, it's **not advised to use this method**. Instead, we should carefully go through each level of the JSON data for **every row** to **ensure safe** handling.

## Question 7/22 *(4 p.)*

Given below are two equivalent pieces of code in Pandas & Apache Spark. Use any one snippet to arrive at the answer. **Select all options that apply.**

**Pandas**

```
df.groupby('SomeColumn').agg(
  meanValue1=('Value1', 'mean'),
  stddevValue1=('Value1', 'std'),
  minValue2=('Value2', 'min'),
  maxValue2=('Value2', 'max')
)
```

**Spark**

```
df.groupBy('SomeColumn').agg(
  F.mean('Value1').alias('meanValue1'),
  F.stddev('Value1').alias('stddevValue1'),
  F.min('Value2').alias('minValue2'),
  F.max('Value2').alias('maxValue2')
)
```

*A.* **Value1** and **Value2** represent a pair of values present in the **SomeColumn** field.

*B.* The result will have **4 columns** not counting **SomeColumn**

*C.* Each row of the result would represent stats for a **distinct** value of the **SomeColumn** field.

*D.* The code computes the **average**, **standard deviation**, **minimum**, and **maximum** for both '**Value1**' and '**Value2**' fields, with each operation being applied to each field.

*E.* The code calculates the **mean** & **standard deviation** for the **first value** in the **SomeColumn** field, and the **min** and **max** for the **second value** in the **SomeColumn** field.

*F.* The code calculates the **mean** & **standard deviation** for the **Value1** field, and the **min** and **max** for the **Value2** field.

## Question 8/22 *(3 p.)*

Given the following **data** and two code snippets using **Pandas** and **Apache Spark** for reshaping and transforming a DataFrame containing student grades, **select all the options that apply.**

```
data = [
    ('A', 80, 90),
    ('B', 70, 60),
    ('C', 85, 75)
]
```

**Pandas**

```
grades_df = pd.DataFrame(
    data=data,
    columns=['StudentID', 'Math', 'Physics']
)
grades_df.melt(
    id_vars=['StudentID'],
    value_vars=['Math', 'Physics'],
    var_name='Subject',
    value_name='Score'
)
```

**SPARK**

```
grades_df = spark.createDataFrame(
    data,
    ['StudentID', 'Math', 'Physics']
)
grades_df.select(
    'StudentID',
    F.expr(
        "stack(2, 'Math', Math, 'Physics', Physics)").alias(
        'Subject', 'Score'
    )
)
```

*A.* The 'Math' and 'Physics' scores are **combined into a single 'Score' column** for each student in the result

*B.* The result will **contain 3 rows**, each representing a single subject's score for each student

*C.* The 'Math' and 'Physics' scores are **combined into a single 'Subject' column** for each student in the result

*D.* The result will **contain 6 rows**, each representing a single subject's score for each student

*E.* The '**melt**' function in the Pandas snippet and the '**stack**' function in the Spark snippet are used to transpose the data

## Question 9/22 *(2 p.)*

After analyzing data from the last ten years, an analyst concludes that the sales of a particular product will **continue to grow** in the next five years. The data shows a consistent **upward trend with no signs of slowing**, and the analyst is assuming this pattern will continue indefinitely. However, no new market strategies or external growth factors are identified to support this future growth. Which underlying **fallacy** might be present in this conclusion? **Select all that apply**

*A.* A **consistent** pattern over time **implies** a **causal** relationship
*B.* **Correlation** between variables **implies** one **causes** the other
*C.* Projecting a past trend infinitely into the future **without** a **solid causal** foundation
*D.* **Excluding** relevant **data** from the analysis

## Question 10/22 *(2 p.)*

A team of data scientists develops a model **predicting a rare event**. During the evaluation phase (once training is complete, and testing has begun), the model's **accuracy** is reported as **99%**. What might be the **best explanation** for this result? **Select all that apply.**

*A.* The evaluation used a **metric** (**accuracy**) that may not be sensitive to the performance on the rare class, causing a **misleading high score**
*B.* The model needs to be **retrained** on a **larger training set** to get a realistic score.
*C.* There's a possibility that the model was tested on a dataset very **similar to the training data.**
*D.* The **data quality** was not properly assessed, and degraded the model's outcomes.
*E.* The model conclusively has **high bias** and **low variance.**

## Question 11/22 *(2 p.)*

You have a DataFrame / Table containing **latitude** and **longitude** columns of locations such that all locations are within a **single city.** Which of the following statements are **true**? **Select all that apply.**

*A.* The **mean** of the latitude and longitude columns returns a close **approximation** of the **geometric centroid** of all points, but not the true value.
*B.* The **median** of the latitude and longitude columns returns a close **approximation** of the **geometric centroid** of all points, but not the true value.
*C.* The mean / median **approximation** gets **closer to the true value** if the **dataset size is increased** to the size of a country instead of a city.
*D.* The reason for the approximation is due to the **curvature of the earth**, which is a non-euclidean shape.
*E.* The **geometric centroid** of all points will always be located **within the physical boundaries of the city**.
*F.* **Skewness** of the latitude and longitude columns has **no bearing** on the **geometric centroid** of the points

## Question 12/22 *(1 p.)*

You are working on an **urban traffic prediction system** to assist drivers with **real-time congestion updates** and alternative route suggestions.
**Rank the following datasets** from most to least **relevant** for the project:
A) Real-time Traffic Data
B) Weather Information
C) Historical Vehicle Sales Data
D) Social Media Trends

*A.* DCBA
*B.* DBAC
*C.* ABCD
*D.* ACBD
*E.* BACD
*F.* BADC

## Question 13/22 *(3 p.)*

You are planning to scrape data from a website that **loads content dynamically using JavaScript.** You notice that regular HTTP requests don't retrieve the required data. What approach would be the **most effective** in this scenario? **Select all that apply**

*A.* Change the **user-agent** in the HTTP request **headers**.
*B.* Convert the website's HTML to **XML** and parse it.
*C.* Use a **headless browser** to execute JavaScript and retrieve content.
*D.* Create a **scraper in the JavaScript language** specifically to tackle this
*E.* Look for JavaScript **generated JSON** objects or **Evaluate JS <script> tags** found in the HTML response

## Question 14/22 *(2 p.)*

You are tasked with **crawling** a large and complex website. The site consists of many **interconnected pages**, and your goal is to traverse them in the most efficient way. What strategy would you likely employ?

*A.* A depth-first search, starting from the homepage
*B.* Scraping the pages in the order they appear in the site's XML sitemap
*C.* Randomly accessing pages without following a specific order
*D.* A breadth-first search, starting from the homepage.

## Question 15/22 *(2 p.)*

You are developing a web crawler to traverse a complex website with a large number of interlinked pages. While traversing the site, you want to ensure that the crawler **does not get stuck in loops** by revisiting pages it has already seen. Which strategy would be **most effective** in ensuring that the **crawler avoids loops and visits each page only once**, considering that the **URLs might have slight variations** (e.g., tracking parameters, fragments) **but still lead to the same content?**

*A.* Employ a depth-first search algorithm. This circumvents the URL issue.
*B.* Hash each visited page's content and look this up while crawling
*C.* Hash each visited url string, and lookup the hash of current and visited URLs when crawling
*D.* Maintain a list of visited URLs with a regex preprocessor and compare each new URL against this list

## Question 16/22 *(1 p.)*

You're creating a modern data pipeline on the cloud that's composed of multiple notebooks or python scripts. How would you **pass data from one step to the other**, as the data travels through the pipeline from start to finish? Choose the **best option** based on ease of **maintainability**, minimizing **development time** and maximizing **modularity.**

*A.* Setup an SSL tunnel between each subsequent pair of tasks on the virtual private cloud, and initiate a data transfer task.
*B.* Use multiple notebooks or scripts for development, but join them together as a single file for production use-cases.
*C.* Run the entire pipeline on an on-demand single cloud machine (VM) and store the files locally at each step. Subsequent tasks would read from the local file system. Terminate the VM when the pipeline is complete.
*D.* Decouple the various tasks by storing data in cloud storage, and have each subsequent task read from cloud storage.

## Question 17/22 *(1 p.)*

Which of the following ideas for **storing data** between various tasks or steps of a modern data pipeline can be considered a **best practice**? A step can be considered as an entire logical component of the pipeline such as "data cleaning", "outlier detection", "data ingestion", etc. Assume that you want to **minimize development time**, **minimize storage costs**, and **maximize support** for handling **complex types** such as JSON or List (Array) in your data.

*A.* Store intermediate results in **normalized relational SQL tables**. Each subsequent task then reads data through a SQL query.
*B.* Store results in a **CSV file** and maintain a neat organization of **directory structure** for CSVs so that appropriate tasks read the right files.
*C.* Store the results in **partitioned parquet files** on cloud storage. Subsequent tasks will read the parquet data output of parent tasks.
*D.* Create a **JSON dump** of each task, and have the subsequent task read the JSON data like **consuming an API.**

## Question 18/22

**Question:**
In a design meeting, there's a debate over using **composition instead of inheritance**. Which of the following scenarios would typically be **more suited for composition?**

**Help:**
Composition and Inheritance are Object Oriented programming principles. View the following examples if you're unfamiliar with these terms.

**Composition Example:**

```
class Arm:
    pass

class Leg:
    pass

class Head:
    pass

class Robot:
    def __init__(self):
        self.arm = Arm()
        self.leg = Leg()
        self.head = Head()
```

**Inheritance Example:**

```
class Robot:
    def __init__(self):
        self.arms = 2
        self.legs = 2

class TalkingRobot(Robot):
    def __init__(self):
        super().__init__()
        self.voice_box = True
```

*A.* When you need to define a clear "is-a" relationship between classes
*B.* When you want to control the access to certain methods of a class
*C.* When you have classes that require multiple inheritance
*D.* When you want to reuse code across several unrelated classes

## Question 19/22 *(2 p.)*

You find that a dataset has several duplicated records. Which of the following would be the **best course of action?**

*A.* Duplicates should always be removed
*B.* Apply data transformations for both duplicated and de-duplicated datasets in parallel for the rest of the pipeline
*C.* Raise an internal issue with the team that the data contains duplicates
*D.* Ascertain if significance is provided to multiple observations of the same entity within the data

## Question 20/22 *(2 p.)*

You want to apply a simple transformation to a numerical column, involving both **squaring** and **logarithmic** operations. Which of the following would provide the **least detrimental performance at scale?**

*A.* **Sort** the column before applying the transformation.
*B.* Create a **custom function** that squares and logs an input value. Then create an **iteration** that loops through each value of the column and applies the function **in sequence.**
*C.* Find a pre-built **vectorized function** through an online search.
*D.* **Divide** the column into smaller chunks. Make a special function that multiplies a cell value by itself and then takes the logarithm. **Use** this **function** in sequence on **each chunk** and then **merge** the results.

## Question 21/22 *(1 p.)*

An elf has the habit of smoking cigarettes. He can make a new cigarette by using **3 cigarette butts**. If he has **27 cigarette butts**, how many cigarettes can he smoke?

*A.* 9
*B.* 12
*C.* 13
*D.* 27
*E.* 81
*F.* 15
*G.* 3
*H.* 0

## Question 22/22 *(2 p.)*

You have a staircase with **256 steps**, and you want to find out the **highest step** from which you can drop a specific type of mobile phone without it breaking. You have more than one phone of the exact same model, and you can drop them as many times as you need to during your tests. **How many trials of phone-drops** do you have to use at a **minimum to discover the step** at which the phone will break for **certain**?

*A.* 256
*B.* 1
*C.* 8
*D.* 2
*E.* 16
*F.* 32
*G.* 31
*H.* 4
*I.* 5