Backend Challenge

PYTHON AND SQL

Helm technical challenge

CHALLENGE

Below are four separate questions relating to Python and SQL. Use your experience and knowledge to answer these questions as best you can. The bonus points are there for you to demonstrate additional related skills for these questions.

BONUS POINTS

These are not required but are typical of the type of knowledge a successful candidate will have or acquire at HELM:

- Version-control your code using git into a github/gitlab repository.
- Test your code's outcomes with appropriate base cases.
- Add additional appropriate testing for your solutions with edge cases.
- Identify solutions that are efficient with respect to either time and/or rows queried.
- Identify when/if scaling is necessary for each and in which ways you would scale your answer (you do not need to code these scaling just explain your reasoning).
- What additions would you make to the sample code and your answer code to make it more robust and/or compliant with modern day best practices for each of these languages?

CHALLENGE 1 - PYTHON

A playlist is considered a repeating playlist if any of the songs contain a reference to (i.e. points to) a previous song in the playlist. Otherwise, the playlist will end with the last song which points to `None`.

Implement a function `is_repeating_playlist` that, efficiently with respect to time used, returns true if a playlist is repeating or false if it is not.

For example, the following code prints "True" as both songs point to each other:

```
first = Song("Hello")
second = Song("Eye of the tiger")

first.next_song(second)
second.next_song(first)
print(first.is_repeating_playlist())
```

Here is some Python code to get you started:

```
class Song:
    def __init__(self, name):
        self.name = name
        self.next = None
```

```
    def next_song(self, song):
        self.next = song

    def is_repeating_playlist(self):
        """
        :returns: (bool) True if the playlist is repeating, False if not.
        """
        # == Add your code here! ==

        return None
```

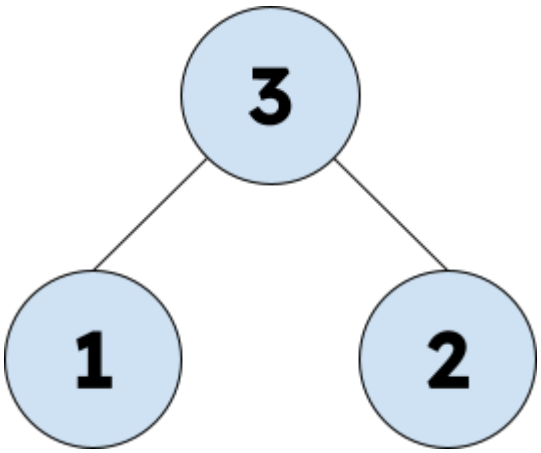
CHALLENGE 2 - PYTHON

A Binary search tree (BST) is a binary tree where the value of each node is larger or equal to the values in all the nodes in that node's left subtree and is smaller than the values in all the nodes in that node's right subtree.

Write a function `def contains(node, value)` that, efficiently with respect to time used, checks if a given binary search tree with root `node` contains the given value.

For example, for the following tree:

```
n1 = Node(value=1, left=None, right=None)
n3 = Node(value=3, left=None, right=None)
n2 = Node(value=2, left=n1, right=n3)
print(contains(n2, 3))
```



Call to `contains(n2, 3)` should return `True` since a tree with root at n2 contains number 3.

Here is some Python code to get you started:

```
import collections
Node = collections.namedtuple('Node', ['left', 'right', 'value'])

def contains(node, value):
        # == Add your code here! ==
        return result
```

CHALLENGE 3 - SQL

A table containing the students enrolled in a yearly course has incorrect data in records with ids between 20 and 100 (inclusive).

```
TABLE enrollments
id INTEGER NOT NULL PRIMARY KEY
year INTEGER NOT NULL
studentId INTEGER NOT NULL
```

Write a query that updates the field `year` of every faulty record to 2015.

CHALLENGE 4 - SQL

App usage data are kept in the following table:

```
TABLE sessions
id INTEGER PRIMARY KEY,
userId INTEGER NOT NULL,
duration DECIMAL NOT NULL
```

Write a query that selects userId and average session duration for each user who has more than one session.

CHALLENGE 5 - DATA PROCESSING

Ingest and process a dataset of your choosing using Python's Pandas library.

Make use of Panda's built-in functions such as group-by, slicing, indexing and conditional filtering to highlight specific features of your dataset (i.e. averages for a particular group in your data, min/max, etc).

BONUS POINTS

- Version-control your code using git into a github/gitlab repository.
- Split your dataset into a training, validation and testing set.
- Build and test a model against this data using Scikit-learn's built-in models.
- Instead build a model using Pytorch or Tensorflow to predict test (unseen) data.