SQL and Final Review

Computer Science Mentors 61A

April 23 to April 25, 2018

Creating Tables, Querying Data

Examine the table, `mentors`, depicted below.

Name	**Food**	**Color**	**Editor**	**Language**
Tiffany	Thai	Purple	Notepad++	Java
Diana	Pie	Green	Sublime	Java
Allan	Sushi	Orange	Emacs	Ruby
Alfonso	Tacos	Blue	Vim	Python
Kelly	Ramen	Green	Vim	Python

1. Create a new table `mentors` that contains all the information above. (You only have to write out the first two rows.)

2. Write a query that lists all the mentors along with their favorite food if their favorite color is green.
```
Diana|Pie
Kelly|Ramen
```

3. Write a query that lists the food and the color of every person whose favorite language is *not* Python.
```
Sushi|Orange
Pie|Green
Thai|Purple
```

4. Write a query that lists all the pairs of mentors who like the same language. (How can we make sure to remove duplicates?)
```
Kelly|Alfonso
Tiffany|Diana
```

Aggregation

CS 61A wants to start a fish hatchery, and we need your help to analyze the data we've collected for the fish populations! Running a hatchery is expensive – we'd like to make some money on the side by selling some seafood (only older fish of course) to make delicious sushi.

The table `fish` contains a subset of the data that has been collected. The SQL column names are listed in brackets.
Table name: `fish`*

Species [species]	**Population [pop]**	**Breeding Rate [rate]**	**$/piece [price]**	**# of pieces per fish [pieces]**
Salmon	500	3.3	4	30
Eel	100	1.3	4	15
Yellowtail	700	2.0	3	30
Tuna	600	1.1	3	20

*(This was made with fake data, do not actually sell fish at these rates)

Hint: The aggregate functions `MAX`, `MIN`, `COUNT`, and `SUM` return the maximum, minimum, number, and sum of the values in a column. The `GROUP BY` clause of a select statement is used to partition rows into groups.

1. Write a query to find the three most populated fish species.

2. Profit is good, but more profit is better. Write a query to select the species that yields the most number of pieces for each price. Your output should include the species, price, and pieces.

3. Write a query to find the total number of fish in the ocean. Additionally, include the number of species we summed. Your output should have the number of species and the total population.

The table `competitor` contains the competitor's `price` for each `species`.

Species [species]	$/piece [price]
Salmon	2
Eel	3.4
Yellowtail	3.2
Tuna	2.6

1. Business is good, but a bunch of competition has sprung up! Through some cunning corporate espionage, we have determined one such competitor's selling prices.

   Write a query that returns, for each species, the difference between our hatchery's revenue versus the competitor's revenue for one whole fish. For example, the table should contain the following row `Salmon|60`.

   Because we make 30 pieces at $4 a piece for $120, whereas the competitor will make 30 pieces at $2 a piece for $60. Finally, the difference is 60.

FINAL REVIEW

Environment Diagrams

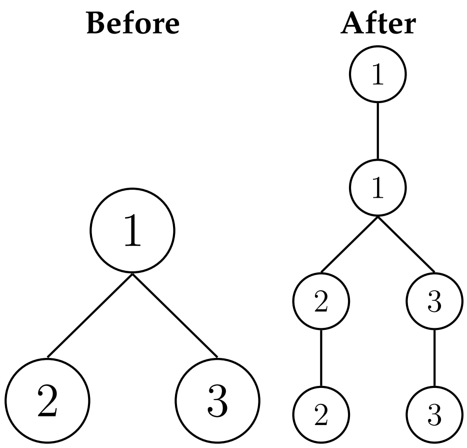
1. Draw the environment diagram for the following code snippet:

```
def one(two):
    three = two
    def four(five):
        nonlocal three
        if len(three) < 1:
            three.append(five)
            five = lambda x: four(x)
        else:
            five = seven + 7
        return five
    two = two + [1]
    seven = 8
    return four(three)

eight = one([])
print(eight(9))
```

Recursive Data Structures

1. DoubleTree hired you to architect one of their hotel expansions! As you might expect, their floor plan can be modeled as a tree and the expansion plan requires doubling each node (the patented double tree floor plan). Here's what some sample expansions look like:



Fill in the implementation for `double_tree`.

```
def double_tree(t):
    """
    Given a tree, return a new tree where entries appear
    twice.
    >>> double_tree(Tree(1))
    Tree(1, [Tree(1)])
    >>> double_tree(Tree(1, [Tree(2), Tree(3)]))
    Tree(1, [Tree(1, [Tree(2, [Tree(2)]),
                      Tree(3, [Tree(3)])
                     ])
            ])
    """
```

2. Fill in the implementation of `double_link`.

```
def double_link(lnk):
    """Using mutation, replaces the second in each pair of
       items
    with the first. The first of each pair stays as is.

    >>> double_link(Link(1, Link(2, Link(3, Link(4)))))
    Link(1, Link(1, Link(3, Link(3))))
    >>> double_link(Link('c', Link('s', Link(6, Link(1,
       Link('a'))))))
    Link('c', Link('c', Link(6, Link(6, Link('a')))))
    """
    if ______________________________________________________:
        return ______________________________________________
    _________________________________________________________
    _________________________________________________________
    return __________________________________________________
```

3. Fill in the implementation of `shuffle`.

```
def shuffle(lnk):
    """Swaps each pair of items in a linked list.

    >>> shuffle(Link(1, Link(2, Link(3, Link(4)))))
    Link(2, Link(1, Link(4, Link(3))))
    >>> shuffle(Link('s', Link('c', Link(1, Link(6,
       Link('a'))))))
    Link('c', Link('s', Link(6, Link(1, Link('a')))))
    """
    if
       _________________________________________________________:
        return ______________________________________________
    front = lnk.rest
    lnk.rest = ______________________________________________
    _________________________________________________________
    return __________________________________________________
```

Scheme

1. Write a Scheme function `insert` that creates a new list that would result from inserting an item into an existing list at the given index. Assume that the given index is between 0 and the length of the original list, inclusive.

   Challenge: Write this as a tail recursive function. Assume append is tail recursive.

```
(define (insert lst item index)

)
```

```
; Tail recursive
(define (insert-tail lst item index)

)
```

Iterators, Generators, and Streams

1. What Would Python Display?

```
class SkipMachine:
    skip = 1
    def __init__(self, n=2):
        self.skip = n + SkipMachine.skip

    def generate(self):
        current = SkipMachine.skip
        while True:
            yield current
            current += self.skip
            SkipMachine.skip += 1

p = SkipMachine()
twos = p.generate()
SkipMachine.skip += 1
twos2 = p.generate()
threes = SkipMachine(3).generate()
```

(a) `next(twos)`

(b) `next(threes)`

(c) `next(twos)`

(d) `next(twos)`

(e) `next(threes)`

(f) `next(twos2)`

2. (a) You and your CS 61A friends are cons. You cdr'd just studied for the final, but instead you scheme to drive away across a stream in a car during dead week. Of course, you would like a variety of food to eat on your road trip.

   Write an infinite stream that takes in a list of foods and loops back to the first food in the list when the list is exhausted.

```
(define (food-stream foods)




)
```

   (b) We discover that some of our food is stale! Every other food that we go through is stale, so put it into a new stale food stream. Assume `is-stale` starts off at 0.

```
(define (stale-stream foods is-stale)




)
```