Creating Tables, Querying Data

Examine the table, mentors, depicted below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Food</th>
<th>Color</th>
<th>Editor</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiffany</td>
<td>Thai</td>
<td>Purple</td>
<td>Notepad++</td>
<td>Java</td>
</tr>
<tr>
<td>Diana</td>
<td>Pie</td>
<td>Green</td>
<td>Sublime</td>
<td>Java</td>
</tr>
<tr>
<td>Allan</td>
<td>Sushi</td>
<td>Orange</td>
<td>Emacs</td>
<td>Ruby</td>
</tr>
<tr>
<td>Alfonso</td>
<td>Tacos</td>
<td>Blue</td>
<td>Vim</td>
<td>Python</td>
</tr>
<tr>
<td>Kelly</td>
<td>Ramen</td>
<td>Green</td>
<td>Vim</td>
<td>Python</td>
</tr>
</tbody>
</table>

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

Solution:
```
create table mentors as
    select 'Tiffany' as name, 'Thai' as food, 'Purple' as color, 'Notepad++' as editor, 'Java' as language
union
    select 'Diana', 'Pie', 'Green', 'Sublime', 'Java' union
    select 'Allan', 'Sushi', 'Orange', 'Emacs', 'Ruby' union
    select 'Alfonso', 'Tacos', 'Blue', 'Vim', 'Python' union
    select 'Kelly', 'Ramen', 'Green', 'Vim', 'Python';
```
2. Write a query that lists all the mentors along with their favorite food if their favorite color is green.

Output:
Diana|Pie
Kelly|Ramen

Solution:
```sql
select m.name, m.food
from mentors as m
where m.color = 'Green';
```

Without aliasing:
```sql
select name, food
from mentors
where color = 'Green';
```

3. Write a query that lists the food and the color of every person whose favorite language is NOT Python.

Output:
Sushi|Orange
Pie|Green
Thai|Purple

Solution:
```sql
select m.food, m.color
from mentors as m
where m.language <> 'Python';
```

Without aliasing:
```sql
select food, color
from mentors
where language != 'Python';
```
4. Write a query that lists all the pairs of mentors who like the same language. (How can we make sure to remove duplicates?)

Output:
Kelly | Alfonso
Tiffany | Diana

Solution:
```
select m1.name, m2.name
from mentors as m1, mentors as m2
where m1.language = m2.language and m1.name > m2.name;
```
2 Fish Population

The 61A mentors want to start a fish hatchery, and they need your help to analyze the data they’ve collected for the fish populations! Also, running a hatchery is expensive – they’d like to make some money on the side by selling some seafood (only older fish of course) to make delicious sushi.

The following table contains a subset of the data that has been collected. The SQL column names are listed in brackets. Note: we must be able to extend your queries to larger tables! (i.e, don’t hard code your answers)

Table name: fish*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Salmon</td>
<td>500</td>
<td>3.3</td>
<td>4</td>
<td>30</td>
</tr>
<tr>
<td>Eel</td>
<td>100</td>
<td>1.3</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>Yellowtail</td>
<td>700</td>
<td>2.0</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>Tuna</td>
<td>600</td>
<td>1.1</td>
<td>3</td>
<td>20</td>
</tr>
</tbody>
</table>

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

5. Aggregation Hint: The aggregate functions \(\text{MAX}, \text{MIN}, \text{COUNT},\) and \(\text{SUM}\) return the maximum, minimum, number, and sum of the values in a column. The \texttt{GROUP BY} clause of a select statement is used to partition rows into groups.

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

**Solution:**
\[
\text{select species from fish order by -pop LIMIT 3;}
\]

(b) 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.

**Solution:**
\[
\text{select species, price, MAX(pieces) from fish GROUP BY price;}
\]
(c) 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.

Solution:
```sql
select COUNT(species), SUM(pop) from fish;
```

(d) Business is good, but a bunch of competition has sprung up! Through some cunning corporate espionage, we have determined that one such competitor plans to open shop with the following rates:

Table name: competitor

<table>
<thead>
<tr>
<th>Species</th>
<th>$/piece</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salmon</td>
<td>2</td>
</tr>
<tr>
<td>Eel</td>
<td>3.4</td>
</tr>
<tr>
<td>Yellowtail</td>
<td>3.2</td>
</tr>
<tr>
<td>Tuna</td>
<td>2.6</td>
</tr>
</tbody>
</table>

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

We make 30 pieces of salmon at $4 for a total revenue of $120, whereas the competitor makes 30 pieces at $2 a piece for a total revenue of $60. The difference is $60. Remember to do this for every species!

Solution:
```sql
select fish.species, (fish.price - competitor.price) * pieces
from fish, competitor
where fish.species = competitor.species;
```

6. **Recursive Select** Suppose these fish breed every day. The population of each fish gets multiplied by its breeding rate every year. Write a recursive select query that creates a table of fish 10 years from now.
Solution:

```sql
with
    annual_pop(yearly_species, yearly_pop, N) as (
        select species, pop, 0 from fish union
        select yearly_species, yearly_pop * rate, N + 1
            from annual_pop, fish
        where yearly_species = species and N < 10
    )
select yearly_species, yearly_pop from annual_pop where N = 10;
```
3 Environment Diagrams

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

```python
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))
```

Solution: https://goo.gl/d71WTd
2. 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:

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>1 1 2 2 3 3</td>
</tr>
</tbody>
</table>

Fill in the implementation for `double_tree`.

```python
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)]]))])
    """
    if t.is_leaf():
        return Tree(t.label, [Tree(t.label)])
    else:
        dbl_branches = [double_tree(c) for c in t.branches]
        return Tree(t.label, [Tree(t.label, dbl_branches)])
```

Solution:
3. Fill in the implementation of `double_link`.

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

>>> double_link(Link(1, Link(2, Link(3, Link(4)))))
Link(1, Link(1, Link(3, Link(4))))

>>> 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 _______________________________
```

**Solution:**

```python
if lst is Link.empty or lst.rest is Link.empty:
    return lst

lst.rest.first = lst.first
double_link(lst.rest.rest)
return lst
```
4. Fill in the implementation of `shuffle`.
   ```python
def shuffle(lst):
    """
    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 ______________________________________
    new_head = lst.rest
    lst.rest = _________________________________________
    return _____________________________________________
```

**Solution:**
```python
if lst == Link.empty or lst.rest == Link.empty:
    return lst
new_head = lst.rest
lst.rest = shuffle(new_head.rest)
new_head.rest = lst
test = new_head
return new_head
```
5. 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.

```scheme
(define (insert lst item index)
  (if (= index 0)
      (cons item lst)
      (cons (car lst) (insert (cdr lst) item (- index 1))))
)
```

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

```scheme
(define (insert lst item index)
  (define (helper lst index so-far)
    (if (or (null? lst) (= index 0))
        (append so-far (cons item lst))
        (helper (cdr lst) (- index 1)
          (append so-far (list (car lst))))
    )
  )
  (helper lst index nil)
)
```
6. Create a `mod_seven` table that has two columns, a number from 0 to 100 and then its value mod 7.

**Hint:** You can create a table first with all of the initial data you will build from, and then build the `mod_seven` table.

**Solution:**

```sql
WITH base(n) AS (  
    SELECT 0 UNION  
    SELECT n+1 FROM base WHERE n+1<7  
),  
mod_seven (n, value) AS (  
    SELECT n, n FROM base UNION  
    SELECT n+7, value FROM mod_seven WHERE n+7<=100  
)  
SELECT * FROM mod_seven;
```

**ALTERNATIVE SOLUTION WITH MODULO OPERATOR**

```sql
WITH mod_seven (n, value) AS (  
    SELECT 0, 0 UNION  
    SELECT n+1, (n+1)%7 FROM mod_seven WHERE n<100  
)  
SELECT * FROM mod_seven;
```

**ALTERNATIVE SOLUTION WITH ONE TABLE**

(This could be a pre-step to approaching the original solution.)

```sql
WITH mod_seven (n, value) AS (  
    SELECT 0, 0 UNION  
    SELECT 1, 1 UNION  
    SELECT 2, 2 UNION  
    SELECT 3, 3 UNION  
    SELECT 4, 4 UNION  
    SELECT 5, 5 UNION  
    SELECT 6, 6 UNION  
    SELECT n+7, value FROM mod_seven WHERE n+7 <= 100  
)  
SELECT * FROM mod_seven;
```
7. Implement `run_length_decoder`, a generator that yields the decoded run length sequence from a list of (value, length) pairs.

```python
def run_length_decoder(encoding):
    
    >>> rld = run_length_decoder([('h', 1), ('e', 1), ('l', 2), ('o', 1)])
    >>> lst(rld)
    ['h', 'e', 'l', 'l', 'o']
```

Solution:

```python
for value, length in encoding:
    for _ in range(length):
        yield value
```
8. (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.

Solution:

\[
\text{(define (food-stream foods)} \\
\quad \text{(cons-stream (car foods)} \\
\quad \quad \text{(food-stream (append (cdr foods)} \\
\quad \quad \quad \text{(list (car foods)))}} \\
\text{)} \]

(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.

Solution:

\[
\text{(define (stale-stream foods is-stale)} \\
\quad \text{(cond ((null? foods) nil)} \\
\quad \quad \text{((= is-stale 1)} \\
\quad \quad \quad \text{(cons-stream (car foods)} \\
\quad \quad \quad \quad \text{(stale-stream (cdr foods) 0)))}} \\
\quad \quad \text{(else (stale-stream (cdr foods) 1)))}} \\
\text{)} \]