LeetCode SQL and Pandas Solutions

Actors and Directors Who Cooperated at Least Three Times

Link: https://leetcode.com/problems/actors-and-directors-who-cooperated-at-least-three-times/

SQL Solution:
```
SELECT actor_id, director_id FROM ActorDirector GROUP BY actor_id, director_id
HAVING COUNT(*) >= 3;
```

Pandas Solution:
```
import pandas as pd def actors_and_directors(actor_director: pd.DataFrame) ->
pd.DataFrame: result = ( actor_director .groupby(['actor_id', 'director_id'])
.size() .reset_index(name='count') .query('count >= 3')[['actor_id', 'director_id']]
) return result
```

Fix Names in a Table

Link: https://leetcode.com/problems/fix-names-in-a-table/

SQL Solution:
```
SELECT user_id, CONCAT(UPPER(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name FROM
Users ORDER BY user_id;
```

Pandas Solution:
```
def fix_names(users: pd.DataFrame) -> pd.DataFrame: users['name'] =
users['name'].str.capitalize() return users.sort_values('user_id')
```

Combine Two Tables

Link: https://leetcode.com/problems/combine-two-tables/

SQL Solution:
```
SELECT p.firstName, p.lastName, a.city, a.state FROM Person p LEFT JOIN Address a ON
p.personId = a.personId;
```

Pandas Solution:
```
def combine_two_tables(person: pd.DataFrame, address: pd.DataFrame) -> pd.DataFrame:
return person.merge(address, on='personId', how='left')[['firstName', 'lastName',
'city', 'state']]
```

Second Highest Salary

Link: https://leetcode.com/problems/second-highest-salary/

SQL Solution:
```
SELECT ( SELECT DISTINCT salary FROM Employee ORDER BY salary DESC LIMIT 1 OFFSET 1 )
AS SecondHighestSalary;
```

Pandas Solution:

```
def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame: salaries =
employee['salary'].drop_duplicates().sort_values(ascending=False) second =
salaries.iloc[1] if len(salaries) >= 2 else None return
pd.DataFrame({'SecondHighestSalary': [second]})
```

List the Products Ordered in a Period

Link: https://leetcode.com/problems/list-the-products-ordered-in-a-period/

SQL Solution:
```
SELECT product_name, SUM(unit) AS unit FROM Orders o JOIN Products p ON o.product_id
= p.product_id WHERE order_date BETWEEN '2020-02-01' AND '2020-02-29' GROUP BY
product_name HAVING SUM(unit) >= 100;
```

Pandas Solution:
```
def products_ordered_in_period(orders: pd.DataFrame, products: pd.DataFrame) ->
pd.DataFrame: feb = orders.query("'2020-02-01' <= order_date <= '2020-02-29'")
result = ( feb.merge(products, on='product_id') .groupby('product_name',
as_index=False)['unit'] .sum() .query('unit >= 100') ) return result
```

Replace Employee ID With The Unique Identifier

Link: https://leetcode.com/problems/replace-employee-id-with-the-unique-identifier/

SQL Solution:
```
SELECT e.unique_id, emp.name FROM Employees emp LEFT JOIN EmployeeUNI e ON emp.id =
e.id;
```

Pandas Solution:
```
def replace_employee_id(employee: pd.DataFrame, employee_uni: pd.DataFrame) ->
pd.DataFrame: return employee.merge(employee_uni, on='id', how='left')[['unique_id',
'name']]
```

Game Play Analysis IV

Link: https://leetcode.com/problems/game-play-analysis-iv/

SQL Solution:
```
SELECT ROUND( COUNT(DISTINCT player_id) / (SELECT COUNT(DISTINCT player_id) FROM
Activity), 2 ) AS fraction FROM Activity WHERE (player_id, DATE_SUB(event_date,
INTERVAL 1 DAY)) IN ( SELECT player_id, MIN(event_date) FROM Activity GROUP BY
player_id );
```

Pandas Solution:
```
def gameplay_analysis(activity: pd.DataFrame) -> pd.DataFrame: first =
activity.groupby('player_id')['event_date'].min().reset_index() next_day =
first.copy() next_day['event_date'] = next_day['event_date'] + pd.Timedelta(days=1)
merged = next_day.merge(activity, on=['player_id', 'event_date'], how='inner')
fraction = round(len(merged['player_id'].unique()) /
len(activity['player_id'].unique()), 2) return pd.DataFrame({'fraction':
[fraction]})
```

Project Employees I

Link: https://leetcode.com/problems/project-employees-i/

SQL Solution:
```
SELECT p.project_id, ROUND(AVG(e.experience_years), 2) AS average_years FROM Project p JOIN Employee e ON p.employee_id = e.employee_id GROUP BY p.project_id;
```

Pandas Solution:
```
def project_employees(project: pd.DataFrame, employee: pd.DataFrame) -> pd.DataFrame: result = ( project.merge(employee, on='employee_id') .groupby('project_id', as_index=False)['experience_years'] .mean() .round(2) .rename(columns={'experience_years': 'average_years'}) ) return result
```

Department Top Three Salaries

Link: https://leetcode.com/problems/department-top-three-salaries/

SQL Solution:
```
SELECT d.name AS Department, e.name AS Employee, e.salary AS Salary FROM Employee e JOIN Department d ON e.departmentId = d.id WHERE ( SELECT COUNT(DISTINCT e2.salary) FROM Employee e2 WHERE e2.departmentId = e.departmentId AND e2.salary > e.salary ) < 3;
```

Pandas Solution:
```
def department_top_three_salaries(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame: merged = employee.merge(department, left_on='departmentId', right_on='id') merged['rank'] = merged.groupby('name_y')['salary'].rank(method='dense', ascending=False) result = merged[merged['rank'] <= 3][['name_y', 'name_x', 'salary']] result.columns = ['Department', 'Employee', 'Salary'] return result
```