

Ruchi Jha

Bootcamp Week 5

1. Actors and Directors Who Cooperated At Least Three Times

SQL-

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

A screenshot of a code editor window. The title bar says '</> Code'. The status bar at the bottom left says 'Saved' and at the bottom right says 'Ln 1, Col 1'. The main area shows the following SQL query:

```
1 SELECT
2   | actor_id,
3   | director_id
4 FROM ActorDirector
5 GROUP BY actor_id, director_id
6 HAVING COUNT(*) >= 3;
7
8
```

The code is syntax-highlighted with blue for keywords like SELECT, FROM, GROUP, and HAVING, and black for identifiers like actor\_id, director\_id, and COUNT(\*). The editor has a dark theme with light-colored text.

Output-

Testcase | [Test Result](#)

**Accepted** Runtime: 76 ms

Case 1

Input

```
ActorDirector =
```

actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5

▽ View more

Output

actor_id	director_id
1	1

Expected

actor_id	director_id
1	1

 Contribute a testcase

Python-

```
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
```

A screenshot of a Jupyter Notebook code cell. The cell title is "Code". The code itself is:

```
</> Code  
Pandas ▾ 🔒 Auto  
1 import pandas as pd  
2  
3 def actors_and_directors(actor_director: pd.DataFrame) -> pd.DataFrame:  
4     result = (  
5         actor_director  
6             .groupby(['actor_id', 'director_id'])  
7             .size()  
8             .reset_index(name='count')  
9             .query('count >= 3')[['actor_id', 'director_id']]  
10    )  
11  
12    return result  
13
```

The cell status bar at the bottom left says "Saved" and at the bottom right says "Ln 13, Col 5".

Output-

Testcase | Test Result

**Accepted** Runtime: 264 ms

Case 1

Input

```
ActorDirector =
```

actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5

View more

Output

actor_id	director_id
1	1

Expected

actor_id	director_id
1	1

## 2. Fix Names in a Table

SQL-

SELECT

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

</> Code

MySQL ▾ 🔒 Auto

```
1 SELECT
2   user_id,
3   CONCAT(UPPER(LEFT(name, 1)), LOWER(SUBSTRING(name, 2))) AS name
4 FROM Users
5 ORDER BY user_id;
6
```

Output-

Testcase | >\_ Test Result

**Accepted** Runtime: 83 ms

Case 1

Input

```
Users =  
| user_id | name |  
| ----- | ----- |  
| 1       | aLice |  
| 2       | b0B  |
```

Output

```
| user_id | name |  
| ----- | ----- |  
| 1       | Alice |  
| 2       | Bob   |
```

Expected

```
| user_id | name |  
| ----- | ----- |  
| 1       | Alice |  
| 2       | Bob   |
```

Pandas-

```
import pandas as pd
```

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

```
</> Code
Pandas ▾ 🔒 Auto

1 import pandas as pd
2
3 def fix_names(users: pd.DataFrame) -> pd.DataFrame:
4     users['name'] = users['name'].str.capitalize()
5     return users.sort_values('user_id')
6
```

Output-

Testcase | > Test Result

**Accepted** Runtime: 283 ms

Case 1

Input

```
Users =
| user_id | name |
| ----- | ----- |
| 1       | aLice |
| 2       | b0B   |
```

Output

```
| user_id | name |
| ----- | ----- |
| 1       | Alice |
| 2       | Bob   |
```

Expected

```
| user_id | name |
| ----- | ----- |
| 1       | Alice |
| 2       | Bob   |
```

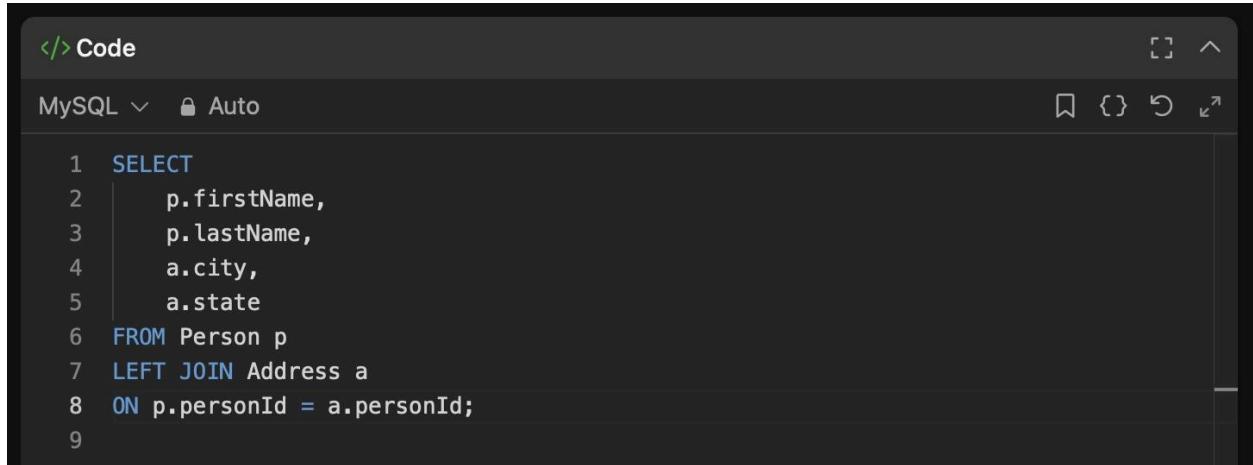
### 3. Combine two tables

SQL-

SELECT

```
p.firstName,  
p.lastName,
```

```
a.city,  
a.state  
FROM Person p  
LEFT JOIN Address a  
ON p.personId = a.personId;
```



A screenshot of a code editor window titled "Code". The editor has a dark theme. In the top left, there's a "MySQL" dropdown and a "Auto" button. The main area contains the following MySQL query:

```
1 SELECT  
2     p.firstName,  
3     p.lastName,  
4     a.city,  
5     a.state  
6 FROM Person p  
7 LEFT JOIN Address a  
8 ON p.personId = a.personId;  
9
```

Output-

Testcase | > Test Result

Accepted Runtime: 91 ms

Case 1

Input

```
Person =  
| personId | lastName | firstName |  
| ----- | ----- | ----- |  
| 1 | Wang | Allen |  
| 2 | Alice | Bob |
```

Address =

```
| addressId | personId | city | state |  
| ----- | ----- | ----- | ----- |  
| 1 | 2 | New York City | New York |  
| 2 | 3 | Leetcode | California |
```

Output

```
| firstName | lastName | city | state |  
| ----- | ----- | ----- | ----- |  
| Allen | Wang | null | null |  
| Bob | Alice | New York City | New York |
```

Expected

```
| firstName | lastName | city | state |  
| ----- | ----- | ----- | ----- |  
| Allen | Wang | null | null |  
| Bob | Alice | New York City | New York |
```

Python-

```
import pandas as pd
```

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

</> Code

Pandas ▾ 🔒 Auto

```
1 import pandas as pd
2
3 def combine_two_tables(person: pd.DataFrame, address: pd.DataFrame) -> pd.
DataFrame:
4     result = person.merge(address, on='personId', how='left')
5     return result[['firstName', 'lastName', 'city', 'state']]
6
```

Output-

Testcase | > Test Result

**Accepted** Runtime: 275 ms

Case 1

Input

Person =		
personId	lastName	firstName
1	Wang	Allen
2	Alice	Bob

Address =			
addressId	personId	city	state
1	2	New York City	New York
2	3	Leetcode	California

Output

firstName	lastName	city	state
Allen	Wang	null	null
Bob	Alice	New York City	New York

Expected

firstName	lastName	city	state
Allen	Wang	null	null
Bob	Alice	New York City	New York

#### 4. Second Highest Salary

SQL-

SELECT

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

</> Code

MySQL ▾ 🔒 Auto

```
1 SELECT
2   (
3     SELECT DISTINCT salary
4       FROM Employee
5         ORDER BY salary DESC
6           LIMIT 1 OFFSET 1
7   ) AS SecondHighestSalary;
8
```

Output-



Testcase



Test Result

Accepted

Runtime: 84 ms

 Case 1 Case 2

Input

Employee =

id   salary	
--   -----	
1	100
2	200
3	300

Output

SecondHighestSalary
-----
200

Expected

SecondHighestSalary
-----
200

Testcase | > Test Result

**Accepted** Runtime: 84 ms

Case 1  Case 2

Input

```
Employee =  
| id | salary |  
| -- | ----- |  
| 1  | 100   |
```

Output

```
| SecondHighestSalary |  
| ----- |  
| null  |
```

Expected

```
| SecondHighestSalary |  
| ----- |  
| null  |
```

Python-

```
import pandas as pd  
  
def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:  
    unique_salaries = employee['salary'].drop_duplicates().sort_values(ascending=False)  
  
    # Check if second highest exists  
    second_highest = unique_salaries.iloc[1] if len(unique_salaries) > 1 else None  
  
    return pd.DataFrame({'SecondHighestSalary': [second_highest]})
```

</> Code

Pandas ▾ 🔍 Auto

```
1 import pandas as pd
2
3 def second_highest_salary(employee: pd.DataFrame) -> pd.DataFrame:
4     unique_salaries = employee['salary'].drop_duplicates().sort_values
5     (ascending=False)
6
7     # Check if second highest exists
8     second_highest = unique_salaries.iloc[1] if len(unique_salaries) > 1 else
9     None
10
11    return pd.DataFrame({'SecondHighestSalary': [second_highest]})
```

Output-

Testcase > Test Result

[ ] ^

Accepted Runtime: 266 ms

Case 1  Case 2

Input

Employee =

id	salary
--	-----
1	100
2	200
3	300



Output

SecondHighestSalary
-----
200

Expected

SecondHighestSalary
-----
200

Testcase | > Test Result

**Accepted** Runtime: 266 ms

Case 1  Case 2

Input

```
Employee =  
| id | salary |  
| -- | ----- |  
| 1  | 100   |
```

Output

```
| SecondHighestSalary |  
| ----- |  
| null    |
```

Expected

```
| SecondHighestSalary |  
| ----- |  
| null    |
```

5. List the Products Ordered in a Period

SQL-

SELECT

```
p.product_name,  
SUM(o.unit) AS unit
```

FROM Products p

JOIN Orders o

ON p.product\_id = o.product\_id

WHERE o.order\_date >= '2020-02-01'

AND o.order\_date < '2020-03-01'

GROUP BY p.product\_name

HAVING SUM(o.unit) >= 100;

Output-

Testcase | > Test Result

Accepted Runtime: 103 ms

Case 1

Input

```
Products =
```

product_id	product_name	product_category
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

```
Orders =
```

product_id	order_date	unit
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80
3	2020-02-17	2
3	2020-02-24	3

View more

Output

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Testcase | [Test Result](#)

2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

Orders =

product_id	order_date	unit
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80
3	2020-02-17	2
3	2020-02-24	3

[View more](#)

Output

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Expected

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Python-

```
import pandas as pd
```

```
def list_products(products: pd.DataFrame, orders: pd.DataFrame) -> pd.DataFrame:
    orders = orders.copy()
    orders['order_date'] = pd.to_datetime(orders['order_date'])

    feb_orders = orders[
        (orders['order_date'] >= '2020-02-01') &
        (orders['order_date'] < '2020-03-01')
    ]

    agg_units = (
```

```

    feb_orders.groupby('product_id', as_index=False)['unit']
        .sum()
)

heavy = agg_units[agg_units['unit'] >= 100]

result = heavy.merge(products[['product_id', 'product_name']], on='product_id', how='inner')

return result[['product_name', 'unit']]

```

**Code**

Pandas Auto

```

1 import pandas as pd
2
3 def list_products(products: pd.DataFrame, orders: pd.DataFrame) -> pd.DataFrame:
4     orders = orders.copy()
5     orders['order_date'] = pd.to_datetime(orders['order_date'])
6
7     feb_orders = orders[
8         (orders['order_date'] >= '2020-02-01') &
9         (orders['order_date'] < '2020-03-01')
10    ]
11
12     agg_units = (
13         feb_orders.groupby('product_id', as_index=False)['unit']
14             .sum()
15     )
16
17     heavy = agg_units[agg_units['unit'] >= 100]
18
19     result = heavy.merge(products[['product_id', 'product_name']],
20                           on='product_id', how='inner')
21
22     return result[['product_name', 'unit']]

```

Output-

Testcase | > Test Result

Accepted Runtime: 272 ms

Case 1

Input

Products =

product_id	product_name	product_category
1	Leetcode Solutions	Book
2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

Orders =

product_id	order_date	unit
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80
3	2020-02-17	2
3	2020-02-24	3

▽ View more

Output

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Testcase | > Test Result

2	Jewels of Stringology	Book
3	HP	Laptop
4	Lenovo	Laptop
5	Leetcode Kit	T-shirt

Orders =

product_id	order_date	unit
1	2020-02-05	60
1	2020-02-10	70
2	2020-01-18	30
2	2020-02-11	80
3	2020-02-17	2
3	2020-02-24	3

▽ View more

Output

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

Expected

product_name	unit
Leetcode Solutions	130
Leetcode Kit	100

## 6. Replace Employee ID With The Unique Identifier

SQL-

SELECT

```
u.unique_id,
e.name
```

FROM Employees e

LEFT JOIN EmployeeUNI u

```
ON e.id = u.id;
```

</> **Code**

MySQL ▾ 🔒 Auto

```
1 SELECT
2     u.unique_id,
3     e.name
4 FROM Employees e
5 LEFT JOIN EmployeeUNI u
6     ON e.id = u.id;
7
8
```

Output-

Testcase | > Test Result

Accepted Runtime: 99 ms

Case 1

Input

Employees =

id	name
--	-----
1	Alice
7	Bob
11	Meir
90	Winston
3	Jonathan

EmployeeUNI =

id	unique_id
--	-----
3	1
11	2
90	3

Output

unique_id	name
-----	-----
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Testcase | >\_ Test Result

90	Winston
3	Jonathan

EmployeeUNI =

id	unique_id
--	-----
3	1
11	2
90	3

Output

unique_id	name
-----	-----
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Expected

unique_id	name
-----	-----
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Python-

import pandas as pd

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

Output-

Testcase | > Test Result

Accepted Runtime: 297 ms

Case 1

Input

Employees =

Employees =	
id   name	
--   -----	
1	Alice
7	Bob
11	Meir
90	Winston
3	Jonathan

EmployeeUNI =

EmployeeUNI =	
id   unique_id	
--   -----	
3	1
11	2
90	3

Output

Output	
unique_id   name	
-----   -----	
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Testcase | > Test Result

90	Winston
3	Jonathan

EmployeeUNI =

id	unique_id
--	-----
3	1
11	2
90	3

Output

unique_id	name
-----	-----
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

Expected

unique_id	name
-----	-----
null	Alice
null	Bob
2	Meir
3	Winston
1	Jonathan

## 7. Game Play Analysis IV

SQL-

WITH first\_login AS (

SELECT player\_id, MIN(event\_date) AS first\_date

FROM Activity

GROUP BY player\_id

),

next\_day\_login AS (

SELECT f.player\_id

FROM first\_login f

JOIN Activity a

ON a.player\_id = f.player\_id

AND a.event\_date = DATE\_ADD(f.first\_date, INTERVAL 1 DAY)

GROUP BY f.player\_id

```
)  
SELECT  
ROUND(  
    (SELECT COUNT(*) FROM next_day_login) * 1.0 /  
    (SELECT COUNT(DISTINCT player_id) FROM Activity)  
, 2) AS fraction;
```

The screenshot shows a MySQL code editor interface. The title bar says '</> Code'. The dropdown menu shows 'MySQL' and 'Auto'. The code area contains the following SQL query:

```
1 WITH first_login AS (  
2     SELECT player_id, MIN(event_date) AS first_date  
3     FROM Activity  
4     GROUP BY player_id  
5 ),  
6 next_day_login AS (  
7     SELECT f.player_id  
8     FROM first_login f  
9     JOIN Activity a  
10    ON a.player_id = f.player_id  
11    AND a.event_date = DATE_ADD(f.first_date, INTERVAL 1 DAY)  
12    GROUP BY f.player_id  
13 )  
14 SELECT  
15 ROUND(  
16     (SELECT COUNT(*) FROM next_day_login) * 1.0 /  
17     (SELECT COUNT(DISTINCT player_id) FROM Activity)  
18 , 2) AS fraction;  
19
```

Output-

Testcase | >\_ Test Result

**Accepted** Runtime: 71 ms

Case 1

Input

Activity =

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output

fraction
-----
0.33

Expected

fraction
-----
0.33

Python-

</> **Code**

Pandas ▾ 🔒 Auto

```
1 import pandas as pd
2
3 def gameplay_analysis(activity: pd.DataFrame) -> pd.DataFrame:
4     df = activity.copy()
5     df['event_date'] = pd.to_datetime(df['event_date'])
6
7     first = (
8         df.groupby('player_id', as_index=False)['event_date']
9             .min()
10            .rename(columns={'event_date': 'first_date'})
11    )
12
13     merged = df.merge(first, on='player_id', how='inner')
14     next_day = merged[merged['event_date'] == (merged['first_date'] + pd.
15     Timedelta(days=1))]
16
17     total_players = first['player_id'].nunique()
18     players_next_day = next_day['player_id'].nunique()
19
20     fraction = round(players_next_day / total_players, 2) if total_players > 0
21     else 0.00
22
23     return pd.DataFrame({'fraction': [fraction]})
```

Output-

Testcase | > Test Result

**Accepted** Runtime: 253 ms

Case 1

Input

Activity =

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

Output

fraction
-----
0.33

Expected

fraction
-----
0.33

8. Project Employees I

SQL-

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;
```

</> Code

MySQL ▾ Auto

```
1 SELECT
2     p.project_id,
3     ROUND(AVG(e.experience_years), 2) AS average_years
4 FROM Project p
5 JOIN Employee e
6     ON p.employee_id = e.employee_id
7 GROUP BY p.project_id;
8 |
```

Output-

Testcase | > Test Result

Accepted Runtime: 88 ms

Case 1

Input

```
Project =
```

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

```
Employee =
```

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output

project_id	average_years
1	2
2	2.5

Expected

project_id	average_years
------------	---------------

Testcase | > Test Result

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee =

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output

project_id	average_years
1	2
2	2.5

Expected

project_id	average_years
1	2
2	2.5

Python-

```
import pandas as pd
```

```
def project_employees_i(project: pd.DataFrame, employee: pd.DataFrame) ->
pd.DataFrame:
    merged = project.merge(employee, on='employee_id', how='inner')

    result = (
        merged.groupby('project_id', as_index=False)['experience_years']
        .mean()
    )

    result['experience_years'] = result['experience_years'].round(2)
```

```
return result.rename(columns={'experience_years': 'average_years'})
```

The screenshot shows a code editor interface with a dark theme. The top bar includes tabs for 'Code' and 'Pandas', and a dropdown for 'Auto'. On the right side of the editor window, there are several icons: a file icon, a class icon, a function icon, a variable icon, and a search icon.

```
</> Code
Pandas ▾ Auto
1 import pandas as pd
2
3 def project_employees_i(project: pd.DataFrame, employee: pd.DataFrame) -> pd.
DataFrame:
4     merged = project.merge(employee, on='employee_id', how='inner')
5
6     result = (
7         merged.groupby('project_id', as_index=False)['experience_years']
8         .mean()
9     )
10
11     result['experience_years'] = result['experience_years'].round(2)
12
13     return result.rename(columns={'experience_years': 'average_years'})
14 |
```

Output-

Testcase | > Test Result

Accepted Runtime: 217 ms

Case 1

Input

Project =

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee =

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output

project_id	average_years
1	2
2	2.5

Expected

Testcase | > Test Result

project_id	employee_id
1	1
1	2
1	3
2	1
2	4

Employee =

employee_id	name	experience_years
1	Khaled	3
2	Ali	2
3	John	1
4	Doe	2

Output

project_id	average_years
1	2
2	2.5

Expected

project_id	average_years
1	2
2	2.5

## 9. Department Top Three Salaries

SQL-

WITH ranked AS (

SELECT

```
d.name AS Department,
e.name AS Employee,
e.salary AS Salary,
DENSE_RANK() OVER (
    PARTITION BY e.departmentId
    ORDER BY e.salary DESC
) AS rnk
```

FROM Employee e

JOIN Department d

ON e.departmentId = d.id

```
)  
SELECT Department, Employee, Salary  
FROM ranked  
WHERE rnk <= 3;  
Output-
```

Testcase | > Test Result

Accepted Runtime: 91 ms

Case 1

Input

Employee =

	id	name	salary	departmentId
	--	-----	-----	-----
	1	Joe	85000	1
	2	Henry	80000	2
	3	Sam	60000	2
	4	Max	90000	1
	5	Janet	69000	1
	6	Randy	85000	1

⋮ View more

Department =

	id	name
	--	-----
	1	IT
	2	Sales

Output

	Department	Employee	Salary
	-----	-----	-----
	IT	Max	90000
	IT	Joe	85000
	IT	Randy	85000
	IT	Will	70000
	Sales	Henry	80000

Testcase | > Test Result

View more

Department =

id   name
--   -----
1   IT
2   Sales

Output

Department   Employee   Salary
-----   -----   -----
IT   Max   90000
IT   Joe   85000
IT   Randy   85000
IT   Will   70000
Sales   Henry   80000
Sales   Sam   60000

Expected

Department   Employee   Salary
-----   -----   -----
IT   Joe   85000
Sales   Henry   80000
Sales   Sam   60000
IT   Max   90000
IT   Randy   85000
IT   Will   70000

Python-

```
import pandas as pd
```

```
def top_three_salaries(employee: pd.DataFrame, department: pd.DataFrame) -> pd.DataFrame:
    merged = employee.merge(
        department,
        left_on='departmentId',
        right_on='id',
        how='inner',
        suffixes=("_dept")
    )

    merged['salary_rank'] = merged.groupby('departmentId')['salary'] \
        .rank(method='dense', ascending=False)
```

```
top3 = merged[merged['salary_rank'] <= 3].copy()

return top3.rename(columns={
    'name_dept': 'Department',
    'name': 'Employee',
    'salary': 'Salary'
})[['Department', 'Employee', 'Salary']]
```

The screenshot shows a Jupyter Notebook cell with the following code:

```
</> Code
Pandas ▾  Auto
1 import pandas as pd
2
3 def top_three_salaries(employee: pd.DataFrame, department: pd.DataFrame) -> pd.
DataFrame:
4     merged = employee.merge(
5         department,
6         left_on='departmentId',
7         right_on='id',
8         how='inner',
9         suffixes=('', '_dept')
10    )
11
12     merged['salary_rank'] = merged.groupby('departmentId')['salary'] \
13     .rank(method='dense', ascending=False)
14
15     top3 = merged[merged['salary_rank'] <= 3].copy()
16
17     return top3.rename(columns={
18         'name_dept': 'Department',
19         'name': 'Employee',
20         'salary': 'Salary'
21     })[['Department', 'Employee', 'Salary']]
22
```

The code defines a function `top_three_salaries` that takes two `pandas.DataFrame` objects, `employee` and `department`, as input. It merges the two DataFrames based on their `departmentId` and `id` columns respectively, using an inner join. The resulting DataFrame is then grouped by `departmentId` and the `salary` column is ranked using the `dense` method, which preserves the rank of tied values. Finally, the function returns a copy of the DataFrame where only the rows with a `salary_rank` of 1 or 2 are retained, and the columns are renamed to `Department`, `Employee`, and `Salary`.

Output-

Testcase | > Test Result

Accepted Runtime: 231 ms

Case 1

Input

Employee =

	id	name	salary	departmentId	
	--	-----	-----	-----	
	1	Joe	85000	1	
	2	Henry	80000	2	
	3	Sam	60000	2	
	4	Max	90000	1	
	5	Janet	69000	1	
	6	Randy	85000	1	



▽ View more

Department =

	id	name	
	--	----	
	1	IT	
	2	Sales	

Output

	Department	Employee	Salary	
	-----	-----	-----	
	IT	Joe	85000	
	Sales	Henry	80000	
	Sales	Sam	60000	
	IT	Max	90000	
	IT	Randy	85000	
	TT	will	70000	

Testcase | >\_ Test Result



▽ View more

Department =

id   name
--   -----
1   IT
2   Sales

Output

Department   Employee   Salary
-----   -----   -----
IT   Joe   85000
Sales   Henry   80000
Sales   Sam   60000
IT   Max   90000
IT   Randy   85000
IT   Will   70000

Expected

Department   Employee   Salary
-----   -----   -----
IT   Joe   85000
Sales   Henry   80000
Sales   Sam   60000
IT   Max   90000
IT   Randy   85000
IT   Will   70000