

Problem Statement

1. Given two tables below, write a query to display the comparison result (higher/lower/same) of the average salary of employees in a department to the company's average salary.

-- Table: salary

id	employee_id	amount	pay_date
1	1	9000	2017-03-31
2	2	6000	2017-03-31
3	2	10000	2017-03-31
4	1	7000	2017-02-28
5	2	6000	2017-02-28
6	2	8000	2017-02-28

-- The employee_id column refers to the employee_id in the following table employee.

employee_id	department_id
1	1
2	2
3	2

-- So for the sample data above, the result is:

pay_month	department_id	comparison
2017-03	1	higher
2017-03	2	lower
2017-02	1	same
2017-02	2	same

2. Write an SQL query to report the students (student_id, student_name) being "quiet" in ALL exams. A "quiet" student is the one who took at least one exam and didn't score neither the high score nor the low score.

-- A "quite" student is the one who took at least one exam and didn't score neither the high score nor the low score.

-- Write an SQL query to report the students (student_id, student_name)

being "quiet" in ALL exams.

-- Don't return the student who has never taken any exam. Return the result table ordered by student_id.

-- The query result format is in the following example.

-- Student table:

```
-- +-----+-----+
-- | student_id | student_name |
-- +-----+-----+
-- | 1          | Daniel      |
-- | 2          | Jade        |
-- | 3          | Stella      |
-- | 4          | Jonathan    |
-- | 5          | Will        |
-- +-----+-----+
```

-- Exam table:

```
-- +-----+-----+-----+
-- | exam_id  | student_id | score  |
-- +-----+-----+-----+
-- | 10       | 1          | 70     |
-- | 10       | 2          | 80     |
-- | 10       | 3          | 90     |
-- | 20       | 1          | 80     |
-- | 30       | 1          | 70     |
-- | 30       | 3          | 80     |
-- | 30       | 4          | 90     |
-- | 40       | 1          | 60     |
-- | 40       | 2          | 70     |
-- | 40       | 4          | 80     |
-- +-----+-----+-----+
```

-- Result table:

```
-- +-----+-----+
-- | student_id | student_name |
-- +-----+-----+
-- | 2          | Jade        |
-- +-----+-----+
```

3. Write a query to display the records which have 3 or more consecutive rows and the number of people more than 100(inclusive).

-- X city built a new stadium, each day many people visit it and the stats are saved as these columns: id, visit_date, people

-- Please write a query to display the records which have 3 or

more consecutive rows and the number of people more than 100(inclusive).

-- For example, the table stadium:

```
-- +-----+-----+-----+
-- | id | visit_date | people |
-- +-----+-----+-----+
-- | 1 | 2017-01-01 | 10     |
-- | 2 | 2017-01-02 | 109    |
-- | 3 | 2017-01-03 | 150    |
-- | 4 | 2017-01-04 | 99     |
-- | 5 | 2017-01-05 | 145    |
-- | 6 | 2017-01-06 | 1455   |
-- | 7 | 2017-01-07 | 199    |
-- | 8 | 2017-01-08 | 188    |
-- +-----+-----+-----+
```

-- For the sample data above, the output is:

```
-- +-----+-----+-----+
-- | id | visit_date | people |
-- +-----+-----+-----+
-- | 5 | 2017-01-05 | 145    |
-- | 6 | 2017-01-06 | 1455   |
-- | 7 | 2017-01-07 | 199    |
-- | 8 | 2017-01-08 | 188    |
-- +-----+-----+-----+
```

-- Note:

-- Each day has only one row record, and the dates are increasing with id increasing.

4. Write an SQL query to find how many users visited the bank and didn't do any transactions, how many visited the bank and did one transaction and so on.

-- Write an SQL query to find how many users visited the bank and didn't do any transactions,

how many visited the bank and did one transaction and so on.

-- The query result format is in the following example:

-- Visits table:

```
-- +-----+-----+
-- | user_id | visit_date |
-- +-----+-----+
-- | 1       | 2020-01-01 |
-- | 2       | 2020-01-02 |
-- | 12      | 2020-01-01 |
-- | 19      | 2020-01-03 |
-- | 1       | 2020-01-02 |
-- | 2       | 2020-01-03 |
-- | 1       | 2020-01-04 |
```

```

-- | 7      | 2020-01-11 |
-- | 9      | 2020-01-25 |
-- | 8      | 2020-01-28 |
-- +-----+-----+
-- Transactions table:
-- +-----+-----+-----+
-- | user_id | transaction_date | amount |
-- +-----+-----+-----+
-- | 1      | 2020-01-02      | 120    |
-- | 2      | 2020-01-03      | 22     |
-- | 7      | 2020-01-11      | 232    |
-- | 1      | 2020-01-04      | 7       |
-- | 9      | 2020-01-25      | 33     |
-- | 9      | 2020-01-25      | 66     |
-- | 8      | 2020-01-28      | 1       |
-- | 9      | 2020-01-25      | 99     |
-- +-----+-----+-----+
-- Result table:
-- +-----+-----+
-- | transactions_count | visits_count |
-- +-----+-----+
-- | 0                  | 4            |
-- | 1                  | 5            |
-- | 2                  | 0            |
-- | 3                  | 1            |
-- +-----+-----+
-- * For transactions_count = 0, The visits (1, "2020-01-01"), (2, "2020-01-02"),
(12, "2020-01-01") and (19, "2020-01-03") did no transactions so visits_count = 4.
-- * For transactions_count = 1, The visits (2, "2020-01-03"), (7, "2020-01-11"),
(8, "2020-01-28"), (1, "2020-01-02") and (1, "2020-01-04") did one transaction so
visits_count = 5.
-- * For transactions_count = 2, No customers visited the bank and
did two transactions so visits_count = 0.
-- * For transactions_count = 3, The visit (9, "2020-01-25")
did three transactions so visits_count = 1.
-- * For transactions_count >= 4, No customers visited the bank and
did more than three transactions so we will stop at transactions_count = 3

```

5. Write an SQL query to generate a report of period_state for each continuous interval of days in the period from 2019-01-01 to 2019-12-31.

-- A system is running one task every day. Every task is independent of the previous tasks. The tasks can fail or succeed.

-- Write an SQL query to generate a report of period_state for each continuous interval of days in the period from 2019-01-01 to 2019-12-31.

-- period_state is 'failed' if tasks in this interval failed or 'succeeded' if tasks

in this interval succeeded. Interval of days are retrieved as start_date and end_date.

-- Order result by start_date.

-- The query result format is in the following example:

-- Failed table:

```
-- +-----+
-- | fail_date |
-- +-----+
-- | 2018-12-28 |
-- | 2018-12-29 |
-- | 2019-01-04 |
-- | 2019-01-05 |
-- +-----+
```

-- Succeeded table:

```
-- +-----+
-- | success_date |
-- +-----+
-- | 2018-12-30 |
-- | 2018-12-31 |
-- | 2019-01-01 |
-- | 2019-01-02 |
-- | 2019-01-03 |
-- | 2019-01-06 |
-- +-----+
```

-- Result table:

```
-- +-----+-----+-----+
-- | period_state | start_date | end_date |
-- +-----+-----+-----+
-- | succeeded    | 2019-01-01 | 2019-01-03 |
-- | failed       | 2019-01-04 | 2019-01-05 |
-- | succeeded    | 2019-01-06 | 2019-01-06 |
-- +-----+-----+-----+
```

-- The report ignored the system state in 2018 as we care about the system in the period 2019-01-01 to 2019-12-31.

-- From 2019-01-01 to 2019-01-03 all tasks succeeded and the system state was "succeeded".

-- From 2019-01-04 to 2019-01-05 all tasks failed and the system state was "failed".

-- From 2019-01-06 to 2019-01-06 all tasks succeeded and the system state was "succeeded".