Average Processing Time per Machine

Objective:
Calculate the average time each machine takes to complete a process based on start and end timestamps.

Tables Involved:

- **Activity**
  - machine_id (int): Identifier for the machine.
  - process_id (int): Identifier for the process.
  - activity_type (enum): Type of activity ('start' or 'end').
  - timestamp (float): Time at which the activity occurred.

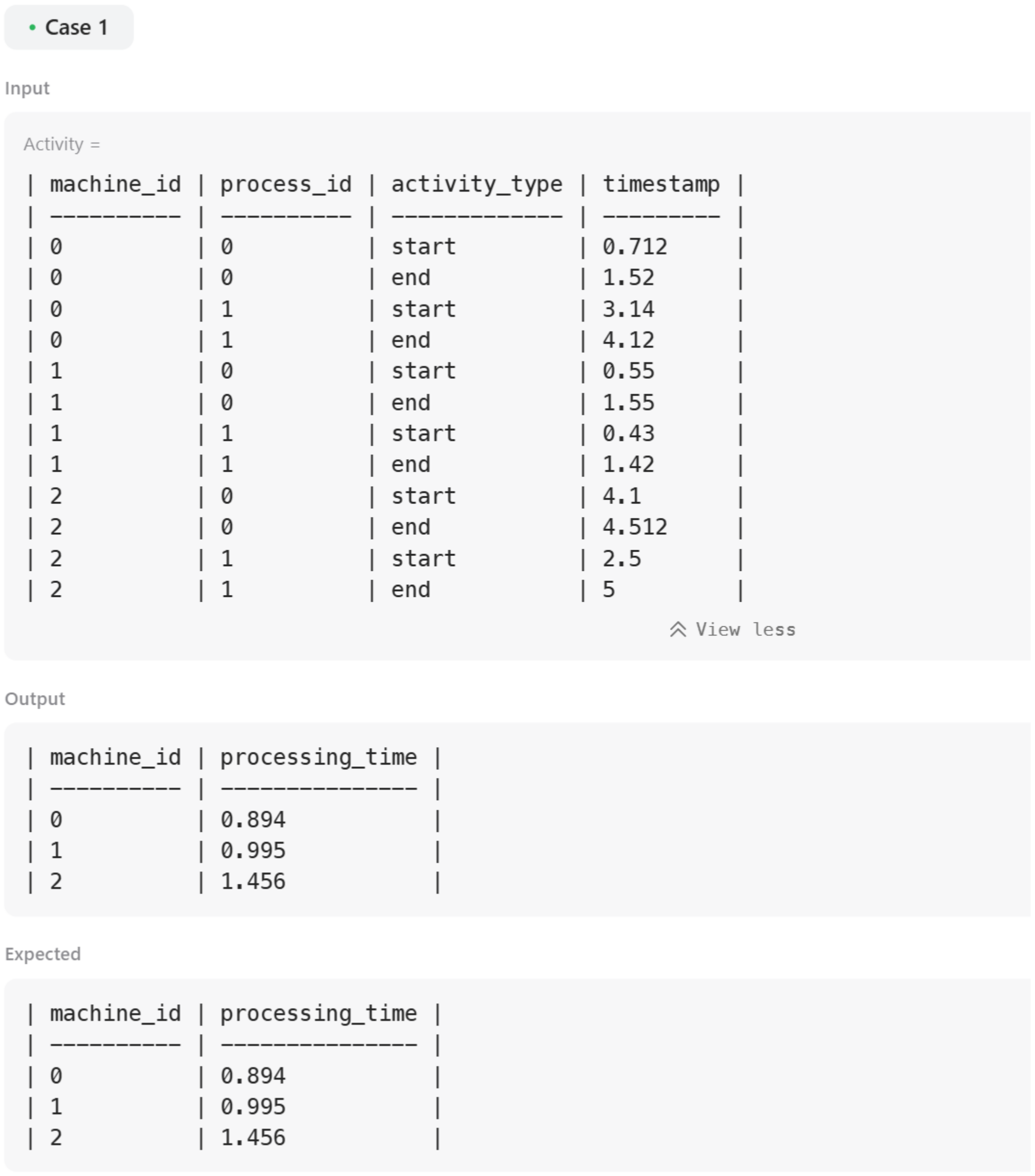
Approach:

1. **Pairing Start and End Timestamps:**
   - Use a self-join on the Activity table to pair each 'start' activity with its corresponding 'end' activity based on machine_id and process_id.
2. **Calculating Processing Time:**
   - For each pair, compute the difference between the 'end' and 'start' timestamps to determine the processing time for each process.
3. **Averaging Processing Times:**
   - Group the results by machine_id and calculate the average processing time for each machine.
4. **Rounding:**
   - Round the average processing time to three decimal places for precision.

SQL Query:

```
SELECT
  start.machine_id,
  ROUND(AVG(end.timestamp - start.timestamp), 3) AS processing_time
FROM
  Activity AS start
  JOIN Activity AS end
```

```sql
  ON start.machine_id = end.machine_id
  AND start.process_id = end.process_id
  AND start.activity_type = 'start'
  AND end.activity_type = 'end'
GROUP BY
  start.machine_id;
```



• Case 1

Input

Activity =

```
| machine_id | process_id | activity_type | timestamp |
| ---------- | ---------- | ------------- | --------- |
| 0          | 0          | start         | 0.712     |
| 0          | 0          | end           | 1.52      |
| 0          | 1          | start         | 3.14      |
| 0          | 1          | end           | 4.12      |
| 1          | 0          | start         | 0.55      |
| 1          | 0          | end           | 1.55      |
| 1          | 1          | start         | 0.43      |
| 1          | 1          | end           | 1.42      |
| 2          | 0          | start         | 4.1       |
| 2          | 0          | end           | 4.512     |
| 2          | 1          | start         | 2.5       |
| 2          | 1          | end           | 5         |
```

View less

Output

```
| machine_id | processing_time |
| ---------- | --------------- |
| 0          | 0.894           |
| 1          | 0.995           |
| 2          | 1.456           |
```

Expected

```
| machine_id | processing_time |
| ---------- | --------------- |
| 0          | 0.894           |
| 1          | 0.995           |
| 2          | 1.456           |
```

Employee Bonus

Objective:
Retrieve the names and bonus amounts of employees who have a bonus less than 1000 or no bonus recorded.

Tables Involved:

1. **Employee**
   - empId (Primary Key): Unique identifier for each employee.
   - name: Name of the employee.
   - supervisor: ID of the employee's supervisor.
   - salary: Salary of the employee.
2. **Bonus**
   - empId (Primary Key, Foreign Key referencing Employee.empId): Unique identifier linking to the Employee table.
   - bonus: Bonus amount received by the employee.

Approach:

- Perform a LEFT JOIN between the Employee and Bonus tables on the empId column to include all employees, regardless of whether they have a corresponding bonus record.Coding Tutorials | Codelabs365
- Filter the results to include only those employees whose bonus is less than 1000 or who do not have a bonus record (i.e., bonus is NULL).

SQL Query:

```
SELECT e.name, b.bonus

FROM Employee e

LEFT JOIN Bonus b ON e.empId = b.empId

WHERE b.bonus < 1000 OR b.bonus IS NULL;
```

• Case 1

Input

Employee =

empId	name	supervisor	salary
3	Brad	null	4000
1	John	3	1000
2	Dan	3	2000
4	Thomas	3	4000

Bonus =

empId	bonus
2	500
4	2000

Output

name	bonus
Brad	null
John	null
Dan	500

Expected

name	bonus
Brad	null
John	null
Dan	500

Counting Student Exam Attendances

Objective:
Determine the number of times each student attended each exam.

Tables Involved:

1. **Students**
   - student_id (Primary key)
   - student_name
2. **Subjects**
   - subject_name (Primary key)
3. **Examinations**
   - student_id
   - subject_name

Approach:

1. **Generate All Student-Subject Combinations:**
   - Perform a CROSS JOIN between the Students and Subjects tables to create all possible student-subject pairs. This ensures that even if a student hasn't attended an exam for a subject, the combination is still represented.
2. **Count Exam Attendances:**
   - Use a LEFT JOIN with the Examinations table on the student_id and subject_name to count the occurrences of each student attending each exam. The COUNT function will tally the number of times each student has attended the exam for a particular subject.
3. **Group and Order Results:**
   - Group the results by student_id, student_name, and subject_name to aggregate the attendance counts.
   - Order the final output by student_id and subject_name for clarity.

SQL Query:

```
SELECT
  s.student_id,
  s.student_name,
  sub.subject_name,
  COUNT(e.student_id) AS attended_exams
FROM
  Students s
CROSS JOIN
  Subjects sub
LEFT JOIN
  Examinations e
ON
  s.student_id = e.student_id
  AND sub.subject_name = e.subject_name
GROUP BY
  s.student_id,
  s.student_name,
  sub.subject_name
ORDER BY
  s.student_id,
  sub.subject_name;
```

Input

Students =

```
| student_id | student_name |
| ---------- | ------------ |
| 1          | Alice        |
| 2          | Bob          |
| 13         | John         |
| 6          | Alex         |
```

Subjects =

```
| subject_name |
| ------------ |
| Math         |
| Physics      |
| Programming  |
```

Examinations =

```
| student_id | subject_name |
| ---------- | ------------ |
| 1          | Math         |
| 1          | Physics      |
| 1          | Programming  |
| 2          | Programming  |
| 1          | Physics      |
| 1          | Math         |
| 13         | Math         |
| 13         | Programming  |
| 13         | Physics      |
| 2          | Math         |
| 1          | Math         |
```

View less

Output

```
| student_id | student_name | subject_name | attended_exams |
| ---------- | ------------ | ------------ | -------------- |
| 1          | Alice        | Math         | 3              |
| 1          | Alice        | Physics      | 2              |
| 1          | Alice        | Programming  | 1              |
| 2          | Bob          | Math         | 1              |
| 2          | Bob          | Physics      | 0              |
| 2          | Bob          | Programming  | 1              |
| 6          | Alex         | Math         | 0              |
| 6          | Alex         | Physics      | 0              |
| 6          | Alex         | Programming  | 0              |
| 13         | John         | Math         | 1              |
| 13         | John         | Physics      | 1              |
| 13         | John         | Programming  | 1              |
```

View less

Expected

```
| student_id | student_name | subject_name | attended_exams |
| ---------- | ------------ | ------------ | -------------- |
| 1          | Alice        | Math         | 3              |
| 1          | Alice        | Physics      | 2              |
| 1          | Alice        | Programming  | 1              |
| 2          | Bob          | Math         | 1              |
| 2          | Bob          | Physics      | 0              |
| 2          | Bob          | Programming  | 1              |
| 6          | Alex         | Math         | 0              |
| 6          | Alex         | Physics      | 0              |
| 6          | Alex         | Programming  | 0              |
| 13         | John         | Math         | 1              |
| 13         | John         | Physics      | 1              |
| 13         | John         | Programming  | 1              |
```

View less

Managers with at Least 5 Direct Reports

Objective:
Identify managers who have five or more direct reports.

Table Involved:

- **Employee**
  - id (Primary key)
  - name (Employee's name)
  - department (Department name)
  - managerId (ID of the employee's manager)

Approach:

1. **Self-Join Method:**
   - Perform a self-join on the Employee table:LinkedIn
     - Join Employee as e with Employee as m on e.managerId = m.id.
   - Group the results by m.id and m.name.
   - Use the HAVING clause to filter groups where the count of e.id is 5 or more.

SQL Query:

```
SELECT m.name

FROM Employee e

JOIN Employee m ON e.managerId = m.id

GROUP BY m.id, m.name

HAVING COUNT(e.id) >= 5;
```

2. **Subquery with WHERE Clause:**
   - Use a subquery to find managerIds with at least five direct reports:DEV Community
     - Select managerId from Employee, group by managerId, and filter with HAVING COUNT(*) >= 5.
   - In the main query, select name from Employee where id is in the list of managerIds obtained from the subquery.

3.

SQL Query:

```
SELECT name
FROM Employee
WHERE id IN (
 SELECT managerId
 FROM Employee
 GROUP BY managerId
 HAVING COUNT(*) >= 5
);
```

3. **Common Table Expression (CTE):**
    - Define a CTE to find managerIds with at least five direct reports:
        - Select managerId from Employee, group by managerId, and filter with HAVING COUNT(*) >= 5.
    - In the main query, select name from Employee where id matches managerId from the CTE.Medium

SQL Query:

```
WITH ManagerCounts AS (
 SELECT managerId
 FROM Employee
 GROUP BY managerId
 HAVING COUNT(*) >= 5
)
SELECT name
FROM Employee
WHERE id IN (SELECT managerId FROM ManagerCounts);
```

Case 1

Input

Employee =

```
| id  | name  | department | managerId |
| --- | ----- | ---------- | --------- |
| 101 | John  | A          | null      |
| 102 | Dan   | A          | 101       |
| 103 | James | A          | 101       |
| 104 | Amy   | A          | 101       |
| 105 | Anne  | A          | 101       |
| 106 | Ron   | B          | 101       |
```

Output

```
| name |
| ---- |
| John |
```

Expected

```
| name |
| ---- |
| John |
```