1. Holiday on Saturday: Continuous Break

Approach

- Check if the requested leave ends on **Friday** and a **holiday falls on the next Saturday**.
- Treat the **Friday-to-Sunday period** as a continuous leave (including the holiday on **Saturday**).

Conditions

- Check if the requested leave end date is **Friday**.
- Check if the **next day (Saturday)** is a holiday.
- Count the days from **Friday to Sunday** as part of the leave period.

Implementation Example

Python Code (Logic)

```
from datetime import datetime, timedelta

# Example holiday list

holidays = ["2025-01-10", "2025-01-17"]  # Add all holiday dates
(e.g., 10th and 17th Jan 2025)



def check_continuous_break(start_date, end_date):

    end_date_obj = datetime.strptime(end_date, "%Y-%m-%d")

    next_day = end_date_obj + timedelta(days=1)

    next_day_str = next_day.strftime("%Y-%m-%d")



    if end_date_obj.weekday() == 3 and next_day_str in holidays:  #
Thursday (3 in Python weekday)

        print(f"Continuous Break: The leave from {end_date} (Thursday)
to Sunday will be treated as a continuous break.")
```

```
        return True

    else:

        print("No holiday on the following Friday. Leave is treated normally.")

        return False



# Example usage

check_continuous_break("2025-01-09", "2025-01-09")  # Outputs: Continuous Break
```

---

2. LOP Amount Calculation:

- **Condition:** If an employee is accepting LOP, calculate the LOP amount based on the employee's **CTC** (Cost to Company) and **current month's days**.
- **Formula:** The LOP amount is calculated as:
  - **Per Day Pay** = `CTC / Current Month's Days`
  - **LOP Amount** = `LOP Days x Per Day Pay`

This formula helps you calculate how much an employee's salary should be reduced for the number of days they are on LOP.

Approach:

1. **Fetch Salary Information:**
   - Retrieve the employee's **CTC** from the **salary table**.
2. **Get Current Month's Days:**
   - Determine the number of days in the current month (for example, 31 days in January, 28/29 in February, etc.).
3. **Calculate Per Day Pay:**
   - **Per Day Pay** = `CTC / Current Month's Days`
4. **Calculate LOP Amount:**
   - If the employee is on **2 days of LOP**, the formula will be:
     LOP Amount = `2 x Per Day Pay`

Example:

Let's say an employee has:

- **CTC** = ₹60,000
- **Current Month's Days** = 30 (for the current month)
- **LOP Days** = 2

1. **Per Day Pay** = ₹60,000 / 30 = ₹2,000
2. **LOP Amount** = 2 x ₹2,000 = ₹4,000

Thus, ₹4,000 would be deducted from the employee's salary as LOP.

Steps in the System:

1. **Retrieve the employee's CTC** from the **salary table**.
2. **Calculate the current month's days** based on the current date.
3. **Apply the LOP formula** to calculate the total LOP deduction based on the number of LOP days.

---

3. LOP Warnings Based on Historical Data

Approach

- Check the **LOP (Loss of Pay)** history of the employee from the `lop` table for past months.
- **Conditions**:
  1. If the employee **previously had LOP**:
     - If **leave balance is available**, show a warning: *"Previously had LOP. Ensure it doesn't repeat to avoid KPI impact."*
     - If **no leave balance**, show a warning: *"You have no leave balance, and LOP will impact your KPI. Make decisions wisely."*
  2. No previous LOP: No warnings needed.

Implementation Example

SQL Query

Fetch LOP History:

```
SELECT COUNT(*) AS lop_count, employee_id

FROM lop_table
```

```sql
WHERE employee_id = 123 AND month >= DATE_SUB(CURDATE(), INTERVAL 6
MONTH)

GROUP BY employee_id;
```

1. **Check Leave Balance**:

```sql
SELECT leave_balance

FROM employee_leave_balance

WHERE employee_id = 123;
```

2. **Python Logic**

```python
def check_lop_warning(employee_id, lop_history, leave_balance):

    if lop_history > 0:  # If employee had LOP previously

        if leave_balance > 0:

            print("Warning: You previously had LOP. Ensure it doesn't
repeat to avoid KPI impact.")

        else:

            print("Warning: You have no leave balance, and LOP will
impact your KPI. Make decisions wisely.")

    else:

        print("No prior LOP. No warnings needed.")



# Example usage

check_lop_warning(employee_id=123, lop_history=2, leave_balance=0)
```

---

4. Categorizing Employees Based on Leave Patterns

Approach

- Analyze monthly leave patterns:
  - **Monthly Leave Taker**: Takes **2 leaves per month**.
  - **Often Leave Taker**: Takes **1 leave per month**.
- Use historical leave data to calculate the monthly leave frequency.

Conditions

1. Query the number of leaves taken by an employee in the past **12 months**.
2. Calculate the **average leaves per month**:
   - = 2 → Monthly Leave Taker.
   - = 1 and < 2 → Often Leave Taker.

Implementation Example

SQL Query

Fetch Leave Count for Last 12 Months:

```
SELECT employee_id, COUNT(*) AS total_leaves

FROM leaves_table

WHERE leave_date >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)

GROUP BY employee_id;
```

1. **Divide Total Leaves by 12 to Get Monthly Average**:

```
     SELECT employee_id, total_leaves / 12 AS avg_leaves_per_month

FROM (

    SELECT employee_id, COUNT(*) AS total_leaves

    FROM leaves_table

    WHERE leave_date >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
    GROUP BY employee_id
) AS leave_data;
```

2. **Python Logic**

```
def categorize_employee_leave(employee_id, total_leaves, months=12):
    avg_leaves = total_leaves / months
    if avg_leaves >= 2:
        category = "Monthly Leave Taker"
    elif avg_leaves >= 1:
        category = "Often Leave Taker"
    else:
        category = "Rarely Leaves"

    print(f"Employee {employee_id} is categorized as: {category}")
    return category

# Example usage
categorize_employee_leave(employee_id=123, total_leaves=24)  #
Outputs: Monthly Leave Taker
```