**1. Date-Based Sorting:**

You can implement sorting based on daily, weekly, or monthly attendance.

**Example Code for Sorting by Date (Repository Layer):**

java

Copy code

// Fetch attendance sorted by date

@Query("SELECT a FROM Attendance a WHERE a.date BETWEEN :startDate AND :endDate ORDER BY a.date ASC")

List<Attendance> findAttendanceByDateRange(@Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

You can modify this query to retrieve:

* **Daily** attendance by specifying a single date as both startDate and endDate.
* **Weekly** or **Monthly** attendance by providing a wider range of dates.

**2. User-Based Sorting:**

To group attendance by user, you can retrieve all attendances of a specific user and sort them by date.

**Example Code for User-Based Sorting (Repository Layer):**

java

Copy code

// Fetch attendance for a specific user sorted by date

@Query("SELECT a FROM Attendance a WHERE a.user.id = :userId ORDER BY a.date ASC")

List<Attendance> findAttendanceByUser(@Param("userId") Long userId);

This will help you calculate:

* **Salary**: Count the total days worked.
* **Overtime**: Identify overtime occurrences.
* **Leave**: Track sick or unpaid leaves taken.

**3. Leave Type Sorting:**

You can categorize attendance based on leave type (sick leave, reserve leave).

**Example Code for Leave Type Sorting (Repository Layer):**

java

Copy code

// Fetch attendance with a specific leave type (paid/unpaid)

@Query("SELECT a FROM Attendance a WHERE a.user.id = :userId AND a.leaveType = :leaveType ORDER BY a.date ASC")

List<Attendance> findAttendanceByLeaveType(@Param("userId") Long userId, @Param("leaveType") LeaveType leaveType);

Here, you can define an enum for LeaveType like:

java

Copy code

public enum LeaveType {

SICK\_PAID, SICK\_UNPAID, RESERVE\_UNPAID

}

This helps separate attendance by leave type and manage salary adjustments accordingly.

**4. Late or Early Check-Ins:**

Tracking late or early check-ins involves comparing the actual check-in time with a predefined work start time.

**Example Code to Track Late or Early Check-Ins (Service Layer):**

java

Copy code

public boolean isLateCheckIn(Attendance attendance, LocalTime standardCheckInTime) {

return attendance.getClockInTime().isAfter(standardCheckInTime);

}

public boolean isEarlyCheckOut(Attendance attendance, LocalTime standardCheckOutTime) {

return attendance.getClockOutTime().isBefore(standardCheckOutTime);

}

You can store a predefined check-in time (e.g., 09:00 AM) and compare it with the actual check-in time for each user:

java

Copy code

LocalTime standardCheckInTime = LocalTime.of(9, 0); // 9:00 AM

LocalTime standardCheckOutTime = LocalTime.of(17, 0); // 5:00 PM

for (Attendance attendance : userAttendances) {

if (isLateCheckIn(attendance, standardCheckInTime)) {

System.out.println("User is late on " + attendance.getDate());

}

if (isEarlyCheckOut(attendance, standardCheckOutTime)) {

System.out.println("User checked out early on " + attendance.getDate());

}

}

**Summary of Actions:**

* **Date-Based Sorting**: Use a date range to fetch and sort attendance.
* **User-Based Sorting**: Fetch user-specific attendance sorted by date.
* **Leave Type Sorting**: Query and sort by leave type to manage paid/unpaid leave.
* **Late or Early Check-Ins**: Compare check-in/check-out times with predefined work hours to flag late/early attendance.

**1. Attendance for Salary Calculation:**

To calculate salary based on attendance and subtract unpaid leaves, we need a query to count the number of present days, paid and unpaid leaves.

**Additional Queries:**

java

Copy code

// Count total present days in a date range (excluding leaves)

@Query("SELECT COUNT(a) FROM Attendance a WHERE a.user.id = :userId AND a.date BETWEEN :startDate AND :endDate AND a.leaveType IS NULL")

long countPresentDays(@Param("userId") Long userId, @Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

// Count unpaid leave days in a date range

@Query("SELECT COUNT(a) FROM Attendance a WHERE a.user.id = :userId AND a.date BETWEEN :startDate AND :endDate AND a.leaveType = 'RESERVE\_UNPAID'")

long countUnpaidLeaveDays(@Param("userId") Long userId, @Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

// Count paid leave days in a date range

@Query("SELECT COUNT(a) FROM Attendance a WHERE a.user.id = :userId AND a.date BETWEEN :startDate AND :endDate AND a.leaveType = 'SICK\_PAID'")

long countPaidLeaveDays(@Param("userId") Long userId, @Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

These queries will allow you to:

* Count **present days** for salary calculation.
* Subtract **unpaid leaves** from the total salary.
* Track **paid leaves** so they don’t affect the salary deduction.

**2. Overtime Tracking:**

You can calculate overtime based on work hours from attendance.

**Additional Query:**

java

Copy code

// Get total overtime hours in a date range

@Query("SELECT SUM(a.overtimeHours) FROM Attendance a WHERE a.user.id = :userId AND a.date BETWEEN :startDate AND :endDate")

Long sumOvertimeHours(@Param("userId") Long userId, @Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate);

If you store overtime hours in the Attendance entity, you can calculate the total overtime for salary purposes.

**3. Performance Tracking:**

Tracking performance based on attendance frequency or late check-ins can help with bonuses or disciplinary actions.

**Additional Query:**

java

Copy code

// Count late check-ins in a date range

@Query("SELECT COUNT(a) FROM Attendance a WHERE a.user.id = :userId AND a.clockInTime > :standardCheckInTime AND a.date BETWEEN :startDate AND :endDate")

long countLateCheckIns(@Param("userId") Long userId, @Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate, @Param("standardCheckInTime") LocalTime standardCheckInTime);

**4. Shift Scheduling:**

You can track which users have been regular in attendance and schedule future shifts accordingly.

**Example Query:**

java

Copy code

// Get users with regular attendance over a date range

@Query("SELECT a.user FROM Attendance a WHERE a.date BETWEEN :startDate AND :endDate GROUP BY a.user.id HAVING COUNT(a) >= :requiredDays")

List<User> findRegularAttendees(@Param("startDate") LocalDate startDate, @Param("endDate") LocalDate endDate, @Param("requiredDays") long requiredDays);

This query returns users who have attended a minimum number of days in a given period.

**Service Layer Implementation:**

In the **service layer**, you can use these queries to implement the logic for salary calculation, performance tracking, and other tasks.

**Example Service Method for Salary Calculation:**

java

Copy code

public double calculateSalary(Long userId, LocalDate startDate, LocalDate endDate) {

double basicSalary = userService.findById(userId).getBasicSalary();

int workingDays = 20; // Assume 20 working days in a month

// Get present days, unpaid leaves, and overtime

long presentDays = attendanceRepository.countPresentDays(userId, startDate, endDate);

long unpaidLeaveDays = attendanceRepository.countUnpaidLeaveDays(userId, startDate, endDate);

long paidLeaveDays = attendanceRepository.countPaidLeaveDays(userId, startDate, endDate);

Long overtimeHours = attendanceRepository.sumOvertimeHours(userId, startDate, endDate);

// Calculate overtime (if true, add one day salary)

double overtimePay = (overtimeHours != null && overtimeHours > 0) ? basicSalary / workingDays : 0;

// Calculate salary

double totalSalary = (basicSalary / workingDays \* presentDays)

+ overtimePay

+ medicalAllowance

- (basicSalary / workingDays \* unpaidLeaveDays)

+ otherAllowances - deductions;

return totalSalary;

}

This service method combines attendance data and leave records to compute the user’s salary, factoring in overtime, unpaid leaves, and paid leaves.