# 📝 Daily Dev Log: Payroll Module 💻

## Day 4: Payroll Logic (SQL Joins for Salary Computation)

Today, I focused on building the **Payroll Module** for my HRMS project. The main goal was to implement the logic for salary computation and to use **SQL JOINs** to enrich payroll records with employee and department details.

### What I Accomplished

* **PayrollDAO Class:** I created a PayrollDAO (Data Access Object) class to handle all database operations related to payroll records. This includes standard CRUD (Create, Read, Update, Delete) methods.
  + A key method I implemented was one that retrieves all payroll records, but with additional information about the employee and their department. This was done using a SQL JOIN query.
* **Payroll Class:** I updated the Payroll class to include fields like employeeName and departmentName. These fields are crucial for storing the data fetched from the JOIN query, allowing the application to display more meaningful information than just an employeeId.
  + To make the class more flexible, I created multiple constructors. For example, one is used for inserting new records, while another is tailored to handle the results from JOIN queries.
* **SQL Joins:** I wrote a SQL query that uses JOIN to combine data from three separate tables: payrolls, employees, and departments. This query allows me to retrieve a comprehensive view of payroll data, including the employee's name and their department's name.

SQL

SELECT

p.id,

e.id AS emp\_id,

e.name AS emp\_name,

d.name AS dept\_name,

p.basic,

p.hra,

p.allowances,

p.deductions,

p.net\_salary,

p.pay\_date

FROM payrolls p

JOIN employees e

ON p.employee\_id = e.id

JOIN departments d

ON e.department\_id = d.id

ORDER BY

p.pay\_date DESC;

* **Queue-based Payroll Processing:** I experimented with using a **LinkedList** as a queue to process multiple payroll records. This **FIFO** (First-In, First-Out) structure ensures that salaries are processed in a specific, sequential order, which is good for batch processing.

### Output Examples

This illustrates the value of using JOIN queries:

* **Without JOIN (just payroll data):** Payroll{id=1, employeeId=101, employeeName='null', departmentName='null', basic=5000.0, hra=2000.0, allowances=1000.0, deductions=500.0, netSalary=7500.0, payDate=2025-09-14}
* **With JOIN (includes employee + department details):** PayrollID=1 | EmpID=101 | Name=John Doe | Dept=IT | NetSalary=7500.00 | Date=2025-09-14

### Key Takeaways

This session was very productive, and I learned several important concepts:

* **Net Salary Calculation:** The formula is straightforward: Net Salary = Basic + HRA + Allowances - Deductions.
* **SQL INNER JOIN:** I now have a solid understanding of how to use INNER JOIN to effectively link related tables and retrieve a single, unified result set.
* **DAO Pattern:** This experience reinforced the importance of the **DAO (Data Access Object)** design pattern, which keeps database-specific code separate from the main business logic.
* **Queues:** I learned how a **queue** can be a powerful tool for managing batch processes, like handling a large number of payroll computations.

### Challenges Encountered

* **Class Mapping:** A key challenge was remembering to correctly map the employeeName and departmentName fields within the Payroll class to the data returned by the JOIN query. Initially, they were left as null.
* **Query Syntax:** Writing the JOIN query was a bit tricky at first, especially getting the ON clause right, but it became much clearer after some practice.
* **Error Handling:** I also worked on properly handling **SQL exceptions** to make the application more robust.