Adding a New Analytics Dashboard to a Spring Boot Application

This tutorial walks through the process of adding a new analytics dashboard to the Spring Boot HR Management Demo application. Specifically, we will implement the "Employees with Long Tenure and Low Salary" query, showing how the original SQL from the `sample_queries.txt` file is integrated directly into the Spring Boot application.

How Spring Boot Uses the Oracle Query

A key feature of the Spring Data JPA framework is its ability to execute database queries directly from repository interfaces without requiring you to write a separate implementation class. This is how the query from the `sample_queries.txt` file is used in the application.

Here's the breakdown of the process:

1. **The Original Query**: The `sample_queries.txt` file provides a collection of useful SQL queries for the HR schema. Query #7, "Employees with Long Tenure and Low Salary," is the one we will use.
2. **Repository Interface and the `@Query` Annotation**: The `EmployeeRepository.java` file is a Java interface that extends Spring Data JPA's `JpaRepository`. Instead of writing a class that implements this interface, we add a method declaration for our query. This method, `findLongTenureLowSalaryRaw()`, is annotated with `@Query`.
3. **Direct Integration**: The SQL code for "Employees with Long Tenure and Low Salary" from the `sample_queries.txt` file is copied directly into the `value` attribute of the `@Query` annotation. The `nativeQuery = true` parameter tells Spring that this is a native SQL query that should be executed directly against the database.
4. **Runtime Magic with Dynamic Proxies**: When the application starts, Spring Data JPA automatically creates a proxy object that implements the `EmployeeRepository` interface. When your code calls the `employeeRepository.findLongTenureLowSalaryRaw()` method, it's actually interacting with this runtime-generated proxy. The proxy takes the SQL string from the `@Query` annotation, executes it, and maps the results to the return type you've specified (in this case, `List<Object[]>`).

This approach allows the exact SQL query to live in the codebase, directly tied to the data access layer, but without the boilerplate of traditional JDBC calls.

Step-by-Step Guide to Adding the New Dashboard

Now, let's add the "Employees with Long Tenure and Low Salary" dashboard to the application.

Step 1: Create the Data Transfer Object (DTO)

First, create a new DTO to hold the results of our query. A DTO is a simple Java object used to transfer data between different layers of the application.

Create a new file named `LongTenureLowSalaryDTO.java` in `src/main/java/com/example/hr/backend_springboot/dto/`:

```
package com.example.hr.backend_springboot.dto;

import java.math.BigDecimal;

public class LongTenureLowSalaryDTO {
    private Long employeeId;
    private String employeeName;
    private String jobTitle;
    private BigDecimal salary;
    private BigDecimal medianSalary;
    private BigDecimal yearsOfService;

    public LongTenureLowSalaryDTO(Number employeeId, String employeeName, String
jobTitle, BigDecimal salary, BigDecimal medianSalary, BigDecimal yearsOfService) {
        this.employeeId = employeeId.longValue();
        this.employeeName = employeeName;
        this.jobTitle = jobTitle;
        this.salary = salary;
        this.medianSalary = medianSalary;
        this.yearsOfService = yearsOfService;
    }

    // Getters and setters...
}
```

Step 2: Update the Employee Repository

Next, add the new query from `sample_queries.txt` to the `EmployeeRepository.java` interface.

```
package com.example.hr.backend_springboot.repository;

import com.example.hr.backend_springboot.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
import org.springframework.data.jpa.repository.Query;
import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

    // ... (other existing queries)

    // Query 7: Employees with Long Tenure and Low Salary
    @Query(value = "WITH JobMedian AS (\n" +
            "    SELECT \n" +
            "        job_id,\n" +
            "        MEDIAN(salary) AS median_salary\n" +
            "    FROM employees\n" +
            "    GROUP BY job_id\n" +
            ")\n" +
            "SELECT \n" +
            "    e.employee_id,\n" +
            "    e.first_name || ' ' || e.last_name AS employee_name,\n" +
            "    j.job_title,\n" +
            "    e.salary,\n" +
            "    jm.median_salary,\n" +
            "    ROUND(MONTHS_BETWEEN(SYSDATE, e.hire_date) / 12, 2) AS
years_of_service\n" +
            "FROM employees e\n" +
            "JOIN jobs j ON e.job_id = j.job_id\n" +
            "JOIN JobMedian jm ON e.job_id = jm.job_id\n" +
            "WHERE MONTHS_BETWEEN(SYSDATE, e.hire_date) / 12 > 5\n" +
            "AND e.salary <= jm.median_salary\n" +
            "ORDER BY years_of_service DESC, salary",
            nativeQuery = true)
    List<Object[]> findLongTenureLowSalaryRaw();
}
```

Step 3: Update the Analytics Controller

Now, add a new request mapping to the `AnalyticsController.java` file. This method will call the repository, process the results into DTOs, and pass them to the view.

```
package com.example.hr.backend_springboot.controller;

import com.example.hr.backend_springboot.dto.*;
import com.example.hr.backend_springboot.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.math.BigDecimal;
import java.util.ArrayList;
```

```java
import java.util.List;

@Controller
public class AnalyticsController {

    @Autowired
    private EmployeeRepository employeeRepository;

    // ... (other existing GET mappings)

    @GetMapping("/analytics/long-tenure-low-salary")
    public String longTenureLowSalary(Model model) {
        List<Object[]> rawResults = employeeRepository.findLongTenureLowSalaryRaw();
        List<LongTenureLowSalaryDTO> tenureData = new ArrayList<>();
        for (Object[] row : rawResults) {
            tenureData.add(new LongTenureLowSalaryDTO(
                (Number) row[0],
                (String) row[1],
                (String) row[2],
                (BigDecimal) row[3],
                (BigDecimal) row[4],
                (BigDecimal) row[5]
            ));
        }
        model.addAttribute("tenureData", tenureData);
        return "analytics/long-tenure-low-salary";
    }
}
```

Step 4: Create the HTML View

Create a new Thymeleaf template to display the results.

Create a file named `long-tenure-low-salary.html` in `src/main/resources/templates/analytics/`:

```html
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Long Tenure & Low Salary</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css" rel="stylesheet">
</head>
<body>
    <div class="container-fluid">
        <nav class="navbar navbar-expand-lg navbar-dark bg-primary">
```

```
            <div class="container-fluid">
                <a class="navbar-brand" href="/analytics/dashboard">
                    <i class="fas fa-chart-line me-2"></i>HR Analytics
                </a>
                <div class="navbar-nav ms-auto">
                    <a class="nav-link" href="/analytics/dashboard">Dashboard</a>
                    <a class="nav-link active" href="/analytics/long-tenure-low-
salary">Long Tenure</a>
                </div>
            </div>
        </nav>

        <div class="container mt-4">
            <div class="d-flex justify-content-between align-items-center mb-4">
                <h1><i class="fas fa-exclamation-triangle text-warning me-2"
></i>Employees with Long Tenure and Low Salary</h1>
                <a href="/analytics/dashboard" class="btn btn-outline-primary">
                    <i class="fas fa-arrow-left me-2"></i>Back to Dashboard
                </a>
            </div>

            <div class="alert alert-warning">
                <i class="fas fa-info-circle me-2"></i>
                This report flags employees with over 5 years of service whose
salaries are at or below the median for their job role.
            </div>

            <div class="card">
                <div class="card-header bg-primary text-white">
                    <h5 class="mb-0"><i class="fas fa-table me-2"></i>Retention Risk
Analysis</h5>
                </div>
                <div class="card-body">
                    <div class="table-responsive">
                        <table class="table table-striped table-hover">
                            <thead class="table-dark">
                                <tr>
                                    <th>Employee ID</th>
                                    <th>Employee Name</th>
                                    <th>Job Title</th>
                                    <th>Salary</th>
                                    <th>Median Salary for Job</th>
                                    <th>Years of Service</th>
                                </tr>
                            </thead>
                            <tbody>
                                <tr th:each="employee : ${tenureData}">
                                    <td th:text="${employee.employeeId}"></td>
                                    <td th:text="${employee.employeeName}"></td>
                                    <td th:text="${employee.jobTitle}"></td>
                                    <td th:text="${'$' +
```

```
#numbers.formatDecimal(employee.salary, 1, 2)}"></td>
                                    <td th:text="${'$' +
#numbers.formatDecimal(employee.medianSalary, 1, 2)}"></td>
                                    <td
th:text="${#numbers.formatDecimal(employee.yearsOfService, 1, 2)}"></td>
                                </tr>
                            </tbody>
                        </table>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></s
cript>
</body>
</html>
```

Step 5: Update the Analytics Dashboard

Finally, add a link to the new report on the main analytics dashboard.

Open `src/main/resources/templates/analytics/dashboard.html` and add the following line to the "Employee Analytics" card:

```
<li><a href="/analytics/long-tenure-low-salary" class="text-decoration-none">
    <i class="fas fa-exclamation-triangle text-warning me-2"></i>Long Tenure, Low
Salary
</a></li>
```

Once you've completed these steps, you can rebuild and run your application. You will find a link to your new dashboard on the main analytics page.