**Practice Java 8**

1. Provide department statistics report: Department Name, start Date, number of Employees, number of female, number of male, number of U23.
2. Get all employee have birth month the same with **input month**
3. Get employee and all their relatives
4. Get employee and 1 emergency contact relatives that have type is FATHER-> MOTHER-> Any
5. List of department and project that they manage (-)
6. List of Project at VIETNAM, + number or employee and total num\_of\_hour
7. List of project at VIETNAM total, num\_of\_hours and Total of salary
8. List of all projects together with managed department name
9. List of all departments together with projects

/\*\*

\* Returns a List of all projects together with managed department name

\* No param

\* Returns: a List of all projects together with managed department name

\*/

@Override

public List<ProjectWithDeptNameDTO> getProjectWithDeptName() {

List<Project> projectList = getAllProjectFromDatabase();

projectList.sort(Comparator.comparing(Project::getName));

return projectWithDeptNameMapper.ProjectsToProjectWithDeptNameDtos(projectList);

}

1. List all projects with number of employees and number of hours in a specific area
2. List all projects with total cost (salary) and hours in specific area

/\*\*

\* Return a List all projects with number of employees and number of hours in a specific area

\* Params: area - the area of the project

\* Returns: a list all projects with number of employees and number of hours in a specific area

\* or empty list if do not have any project in this area.

\*/

@Override

public List<ProjectWithNumOfHourAndNumOfEmplDTO> getAllProjectWithEffort(String area) {

// Using Native Query

// List<Object[]> resultList = em.createNativeQuery("select p.id,p.name,p.area,p.managed\_department," +

// " count(employee\_Id) as numOfEmployees, sum(number\_of\_hour) as numOfHours" +

// " from isc.project p, isc.assignment a" +

// " where a.project\_id=p.id" +

// " group by p.id,p.name,p.area,p.managed\_department").getResultList();

//

// return resultList.stream().map(r -> {

// ProjectWithNumOfHourAndNumOfEmplDTO projectWithEffort = new ProjectWithNumOfHourAndNumOfEmplDTO();

// projectWithEffort.setId((Integer) r[0]);

// projectWithEffort.setName((String) r[1]);

// projectWithEffort.setArea((String) r[2]);

// projectWithEffort.setDepartmentId((Integer) r[3]);

// projectWithEffort.setNumOfEmployees(((BigInteger) r[4]).intValue());

// if (Objects.nonNull(r[5])) {

// projectWithEffort.setNumOfHours(((BigDecimal) r[5]).intValue());

// }

// return projectWithEffort;

// }).collect(Collectors.toList());

return em.createQuery("from Assignment", Assignment.class).getResultList().stream()

.collect(Collectors.groupingBy(Assignment::getProject))

.entrySet()

.stream()

.filter(entry -> entry.getKey().getArea().equals(area))

.map(projectEntry -> {

Project project = projectEntry.getKey();

List<Assignment> assignmentList = projectEntry.getValue();

ProjectWithNumOfHourAndNumOfEmplDTO projectWithEffort = new ProjectWithNumOfHourAndNumOfEmplDTO();

projectWithEffort.setId(project.getId());

projectWithEffort.setName(project.getName());

projectWithEffort.setArea(project.getArea());

projectWithEffort.setDepartmentId(project.getDepartment().getId());

projectWithEffort.setNumOfEmployees(assignmentList.size());

int numOfHours = (assignmentList.stream().mapToInt(Assignment::getNumberOfHour)).sum();

projectWithEffort.setNumOfHours(numOfHours);

return projectWithEffort;

}).collect(Collectors.toList());

}

/\*\*

\* Returns a List all projects with total cost (salary) and hours in specific area

\* Params: area - the area of the project

\* Returns: a List all projects with total cost in specific area

\* or empty list if there has no project in this area.

\*/

@Override

public List<ProjectWithHoursAndCostDTO> getAllProjectWithHoursAndCost(String area) {

return em.createQuery("from Assignment", Assignment.class).getResultList().stream()

.collect(Collectors.groupingBy(Assignment::getProject))

.entrySet()

.stream()

.filter(entry -> entry.getKey().getArea().equals(area))

.map(projectEntry -> {

Project project = projectEntry.getKey();

List<Assignment> assignmentList = projectEntry.getValue();

ProjectWithHoursAndCostDTO projectWithCostDTO = new ProjectWithHoursAndCostDTO();

projectWithCostDTO.setId(project.getId());

projectWithCostDTO.setName(project.getName());

projectWithCostDTO.setArea(project.getArea());

projectWithCostDTO.setDepartmentId(project.getDepartment().getId());

projectWithCostDTO.setHours(assignmentList.stream().mapToInt(Assignment::getNumberOfHour).sum());

double cost = assignmentList.stream().mapToDouble(a -> a.getEmployee().getSalary() \* 1000 / TOTAL\_HOURS\_PER\_MONTH \* a.getNumberOfHour()).sum();

projectWithCostDTO.setCost(String.format("%.2f", cost));

return projectWithCostDTO;

}).collect(Collectors.toList());

}

private List<Project> getAllProjectFromDatabase() {

return em.createQuery("from Project", Project.class).getResultList();

}

1. list of employees have not being assigned in any project

Example:

/\*\*

\* returns list of employees have not being assigned in any project

\* No param

\* Returns: a list of employees have not being assigned in any project ar empty list if all employees have been assigned

\*/

public List<EmployeeDTO> getAllEmployeesNotAssigned() {

List<Integer> employeeInProject = assignmentService.getAssignmentsFromDatabase()

.stream()

.map(a -> a.getEmployee().getId())

.distinct()

.collect(Collectors.toList());

return getAllEmployees()

.stream()

.filter(e -> !employeeInProject.contains(e.getId()))

.collect(Collectors.toList());

}

1. List of employee work in project which has been managed by another department

* Example:

/\*\*

\* Returns a List of employee work in project which has been managed by another department

\* No param

\* Returns: a List of employee work in project which has been managed by another department

\* or empty list if all employees just work for their department

\* \*/

public List<EmployeeProjectDTO> getEmployeesWorkInOtherDepartment() {

return assignmentService.getAssignmentsFromDatabase()

.stream()

.collect(Collectors.groupingBy(Assignment::getEmployee))

.entrySet()

.stream()

.filter(employeeListEntry->employeeListEntry.getValue()

.stream()

.map(a->a.getProject().getDepartment().getId())

.collect(Collectors.toList())

.stream()

.anyMatch(departmentId-> !Objects.equals(departmentId, employeeListEntry.getKey().getDepartment().getId())))

.map(entry -> {

EmployeeDTO employeeDTO = employeeMapper.EmployeeToEmployeeDto(entry.getKey());

List<ProjectDTO> projectDTOS = projectMapper.ProjectsToProjectDtos(entry.getValue()

.stream()

.map(Assignment::getProject)

.collect(Collectors.toList()));

return new EmployeeProjectDTO(employeeDTO, projectDTOS);

}).collect(Collectors.toList());

}