//Job Search History

```
match (e0: Employee {employeeID: 22})-[r0:SEARCHED_FOR]->(:Job)<-[r1:SEARCHED_FOR]-(e1: Employee)
match
        (e0)-[:SEARCHED_FOR]->(j0:Job),
        (e1)-[:SEARCHED_FOR]->(j1:Job)
with
        e0,e1,
        count (distinct j0) as j0Count, count (distinct j1) as j1Count
match (e0)-[:SEARCHED_FOR]->(j:Job)<-[:SEARCHED_FOR]-(e1)
with
        e0,e1,
        j0Count, j1Count, count (j) as commonJobSearchedCount
where
        id(e0) < id(e1) and
        commonJobSearchedCount / 0.2 >= j1Count
return e0, e1
order by j0Count, j1Count
```

// Employees' connection preferences

```
call{match (e0: Employee{employeeID:'428'})-[r:CONNECTED_TO]->(e1:Employee)
with toFloat(r.score) as scores, e1 as knownEmployees
order by scores desc limit 5
return collect(knownEmployees) as knownEmployees}

match (e:Employee)-[r:WORKED_AT]->(j:Job)
where e in knownEmployees
return j
```

// Connecting employees with same previous jobs

```
call{match (e:Employee)-[r:WORKED_AT]->(j:Job)
with e as employee, j as jobs, count(r) as associations
order by associations desc limit 5
return collect(employee) as experiencedEmployees}

match (e0: Employee)-[:CONNECTED_TO]->(e1:Employee)
where e0 in experiencedEmployees
return e0, e1


// Matching jobs with the characteristics of employees

call{Match (e0: Employee {employeeID:'390'})
return apoc.convert.fromJsonList(e0.technicalSkills) as skills, e0.location as loc, e0.degree as deg}

match (j: Job)
with  j as jobs, size(apoc.coll.intersection(skills, apoc.convert.fromJsonList(j.technicalSkills)))*2 + size(apoc.coll.intersection(skills,
apoc.convert.fromJsonList(j.preferredSkills))) as skillscore
where jobs.location = loc  and jobs.degree = deg
return jobs
order by skillscore DESC LIMIT 10
```