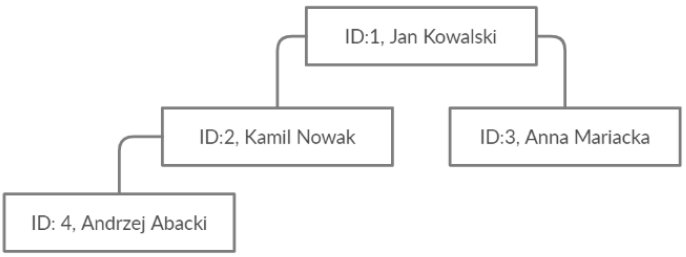
1. W implementowanym systemie przechowujemy informację o strukturze pracowników. Każdy z nich może mieć przypisanego przełożonego. Poniższa klasa obrazuje aktualny stan w aplikacji:

```
public class Employee {
        public int Id { get; set;}
        public String Name { get; set;}
        public int? SuperiorId { get; set;}
        public virtual Employee Superior { get; set;}
}
```

W celu optymalizacji dostałeś zadanie, aby utworzyć strukturę która przechowuje informację o wszystkich poziomach przełożonego.



(Przykład :
Jan Kowalski jest przełożonym **Kamila Nowaka**. **Kamil Nowak** jest przełożonym **Andrzeja Abackiego**. Oznacza to że **Jan Kowalski** jest przełożonym **Andrzeja Abackiego** rzędu 2).

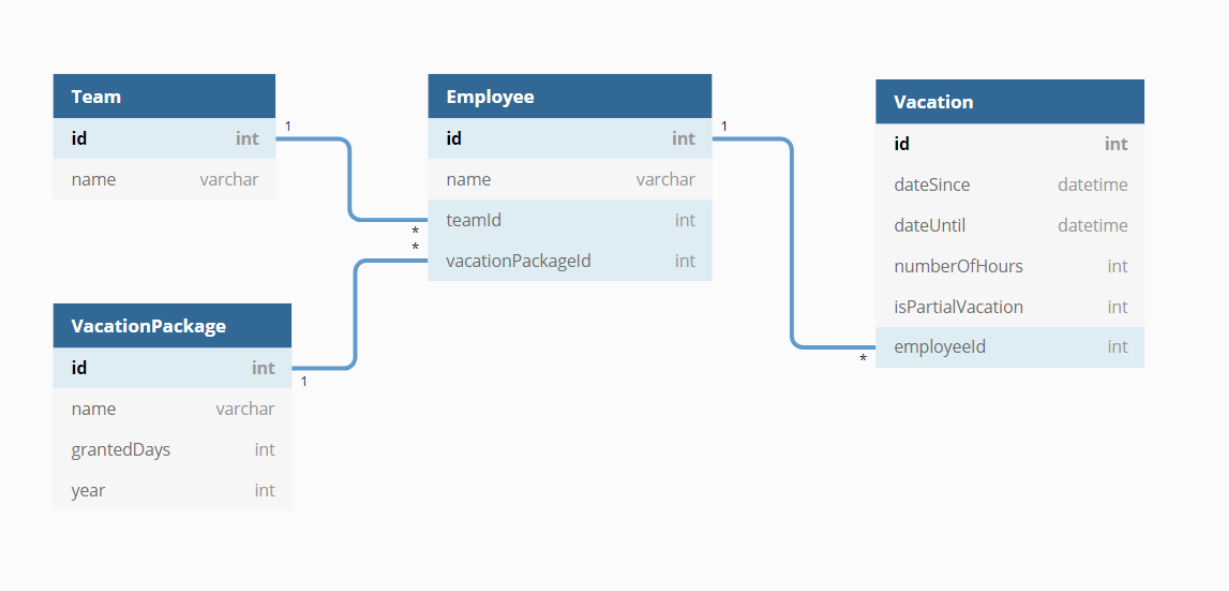
Zaproponowane rozwiązanie powinno pozwalać na określenie czy dany pracownik jest przełożonym dowolnego rzędu dla drugiego pracownika oraz przechowywać liczbę tego rzędu.
Dodaj odpowiednie klasy oraz sposób ich wypełnienia na podstawie listy wszystkich pracowników:

```
// Przykładowa klasa która pozwoli przechowywać relację oraz rząd relacji
// pomiędzy pracownikami
public class EmployeesStructure {
}

// Przykładowa metoda do implementacji która wypełnia zaproponowaną
// strukturę danych
public List<EmployeeStructure > FillEmployeesStructure(List <Employees> employees)

// Przykładowa metoda do implementacji, która zwraca rząd przełożonego
// lub null kiedy superior nie jest przełożonym employee
public int? GetSupieriorRowOfEmployee (int employeeId, int superiorId)
```

Przykład wywołania:

```
var row1 = GetSuperiorRowOfEmployee(2,1); // row1 = 1
var row2 = GetSuperiorRowOfEmployee(4,3); // row2 = null
var row3 = GetSuperiorRowOfEmployee(4,1); // row3 = 2
```

2. Dany jest schemat bazy danych:



Employee { Id, Name, TeamId, PositionId, VacationPackageId }
Team { Id, Name }
VacationPackage { Id, Name, GrantedDays, Year }
Vacations { Id, DateSince, DateUntil, NumberOfHours, IsPartialVacation, EmployeeId }

Napisz zapytanie za pomocą **SQL** lub **LINQ EF**, które:

a) zwraca listę wszystkich pracowników z zespołu o nazwie ".NET", którzy mają co najmniej jeden wniosek urlopowy w 2019 roku.
b) zwraca listę pracowników wraz z liczbą dni urlopowych zużytych w bieżącym roku (za dni zużyte uznajemy wszystkie dni we wnioskach urlopowych które są w całości datą przeszłą).
c) zwraca listę zespołów w których pracownicy nie złożyli jeszcze żadnego dnia urlopowego w 2019 roku.

3. Uzupełnij metodę w języku C# która wylicza ile jeszcze dni urlopowych ma do wykorzystania pracownik w bieżącym roku. (Struktura danych jak w zadaniu 2)

```
public int CountFreeDaysForEmployee(Employee employee, List<Vacation> vacations,
VacationPackage vacationPackage )
{

}
```

4. Na podstawie zadania 3 napisz metodę która sprawdza czy pracownik może zgłosić wniosek urlopowy. (Struktura danych jak w zadaniu 2)

```
public bool IfEmployeeCanRequestVacation(Employee employee, List<Vacation> vacations, VacationPackage vacationPackage)
{

}
```

5. Napisz dwa proste testy jednostkowe które sprawdzą działanie zaimplementowanej metody z zadania numer 4.

```
[Test]
public void employee_can_request_vacation()
{
}

[Test]
public void employee_cant_request_vacation()
{
}
```

6. Zakładając, że parametry metody z zadania 3 pobieramy bezpośrednio z bazy danych. Czy znasz jakieś sposoby na optymalizację liczby zapytań SQL w tego typu przypadkach? Wymień i opisz krótko każdy z nich.