# Uitwerkingen subtyperingen

In dit document beschrijf ik de vier verschillende manieren waarop ik de subtyperingen in mijn EER-model kan uitwerken in mijn DBO.

## Subtypering 1

**Employee**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | nvarchar |
| Function | nvarchar |
| TeamLeaderId | Int (Kan leeg zijn) |

TeamleaderId geldt alleen voor Employee met Function ProductionWorker of TeamLeader. Deze kan dus leeg zijn. Alle andere relaties tussen Employee en andere entiteiten zijn allemaal koppeltabellen. Dus er komen geen extra velden in de tabel Employee.

## Subtypering 2

Deze is hetzelfde als manier 1, maar dan in plaats van Function als nvarchar in de database op te slaan, krijg je een aparte tabel met de verschillende functies.

**Employee**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | nvarchar |
| Function | int |
| TeamLeaderId | Int (Kan leeg zijn) |

**Function**

|  |  |
| --- | --- |
| Id | int |
| Name | nvarchar |

## Subtypering 3

Bij deze manier krijgt elke variatie van Employee een eigen tabel. Het nadeel hier is dat alle generieke gegevens in elke tabel weer terugkomen. De enigste tabel waar een extra veld bijkomt is EmployeeProductionWorker, voor de rest zijn alle tabellen hetzelfde.

**EmployeeHRM**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | nvarchar |

**EmployeeSales**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | nvarchar |

**EmployeeProductionWorker**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | nvarchar |
| TeamLeaderId | Int (Kan leeg zijn) |

**EmployeeTeamLeader**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | Nvarchar |

## Subtypering 4

Deze manier lijkt op de derde manier. Maar in plaats van de generieke gegevens in elke variatietabel terug te laten komen, wordt er vanuit de variatietabel verwezen naar de Employee tabel. Hier staan alle generieke gegevens in.

**Employee**

|  |  |
| --- | --- |
| Id | int |
| FirstName | nvarchar |
| LastName | nvarchar |

**EmployeeHRM**

|  |  |
| --- | --- |
| Id | int |
| EmployeeId | int |

**EmployeeSales**

|  |  |
| --- | --- |
| Id | int |
| EmployeeId | int |

**EmployeeProduction**

|  |  |
| --- | --- |
| Id | int |
| EmployeeId | int |
| TeamLeaderId | Int (Kan leeg zijn) |

**EmployeeTeamLeader**

|  |  |
| --- | --- |
| Id | int |
| EmployeeId | int |

## Verschillen

Wat ik hier heb beschreven, is het verschil in de velden in deze tabellen. Er zijn wel nog relaties die niet als velden in deze tabellen zijn vastgelegd, maar wel in koppeltabellen.

Het voordeel van manier 1 en 2 is dat als ik het type van de werknemer verander, niet een heel record van een tabel naar de andere hoef te verplaatsen.

Het nadeel van manier 1 en 2 is dat er een leeg veld TeamLeaderId kan zijn. Maar het is er wel maar één.

Het nadeel van manier 3 en 4 is dat er voor elk type Employee, er een extra tabel is. Wat het DBO weer complexer maakt. Het zijn er tot nu toe 4.

Het nadeel van manier 3 is dat alle generieke velden, 8 velden, in elke variatietabel terugkomt.

Het nadeel van manier 3 en 4 is ook dat wanneer elke Employee opgevraagd moet worden, alle variatietabellen gejoind moeten worden.

## Conclusie

Manier 1 spreekt mij het meeste aan, omdat het enigste nadeel hiervan is dat ik mogelijk één leeg veld TeamLeaderId heb.

Manier 2 vind ik zelf niet zo mooi, omdat er een discriminatortabel komt met alleen een id-veld en een functie-veld. Deze manier zou handig zijn geweest als de functietabel meerdere velden had. Of als een Employee meerdere functies kon hebben, waardoor je weer een koppeltabel krijgt tussen Employee en de discriminatortabel.

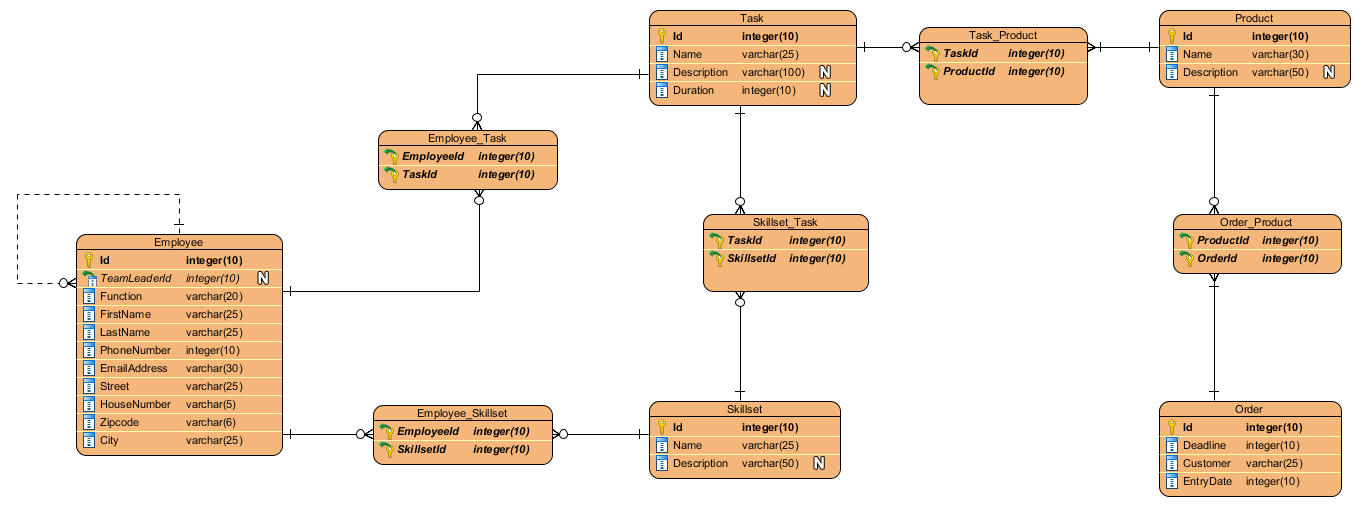
Manier 3 lijkt mij niet handig, omdat ik extra tabellen krijg met generieke gegevens. En alleen in EmployeeProductionWorker staat er één extra veld bij.

Manier 4 heeft hetzelfde probleem, maar dan staan er geen generieke gegevens in elke variatietabel, maar een foreignkey naar de generieke Employee tabel. Hierdoor heb ik dan 3 tabellen met alleen een Id en een foreignkey naar de generieke Employee tabel.

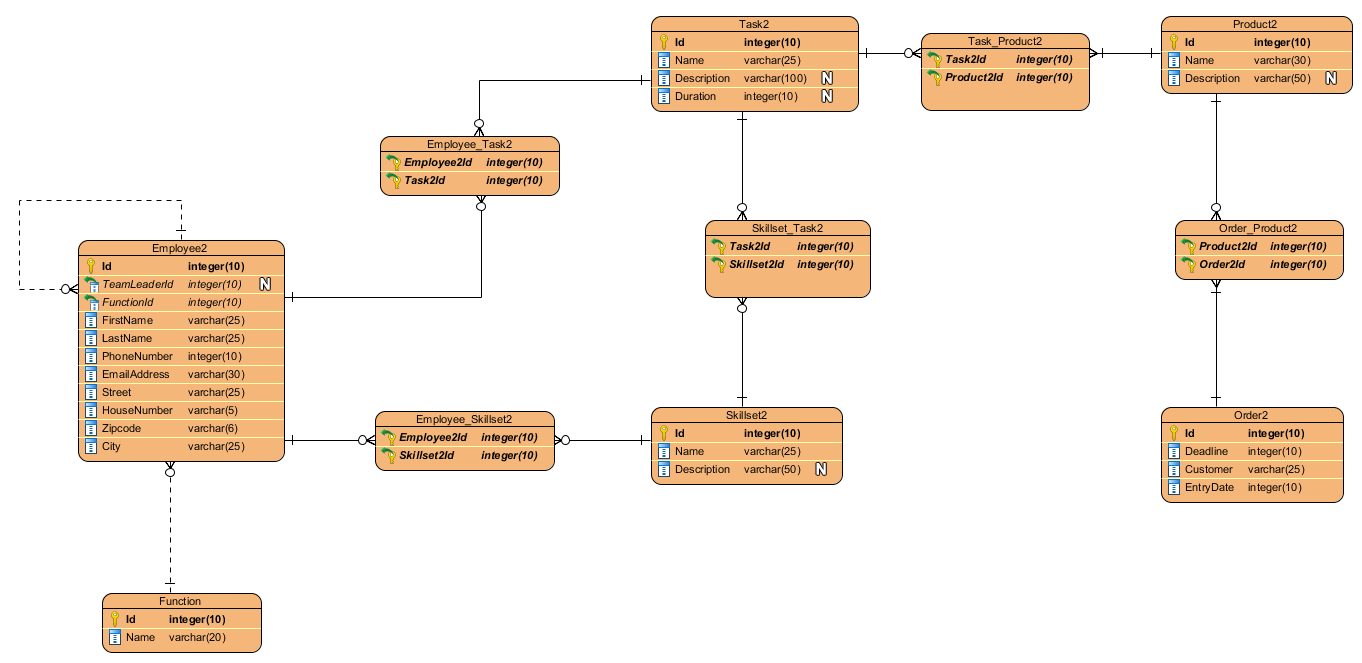
Dus ik ga manier 1 uitwerken.

Hierbij is er nog wel een nadeel dat wanneer je een de status van Employee verandert, er eventueel nog relaties zijn naar entiteiten waar die Employee niks meer mee te maken heeft. Bijvoorbeeld als je een ProductionWorker naar SalesManager verandert, dat die nog een relatie met een taak heeft. Dit kan ik oplossen met triggers en constraints.

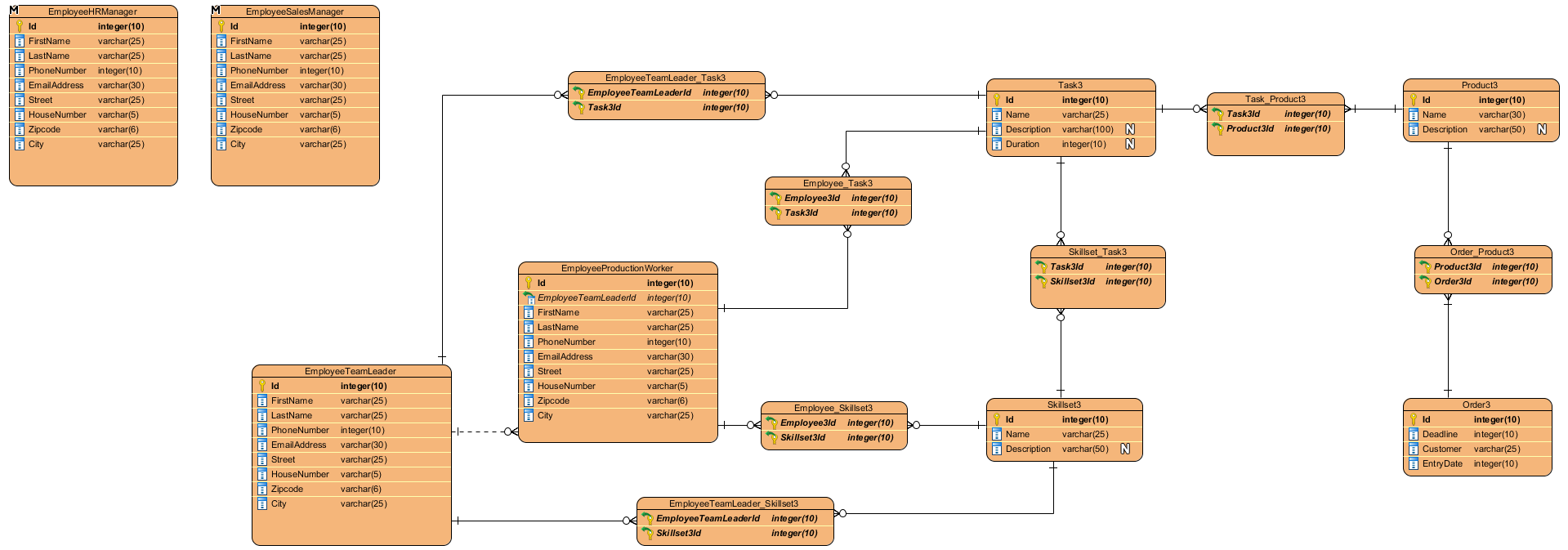
## DBO: Subtypering 1



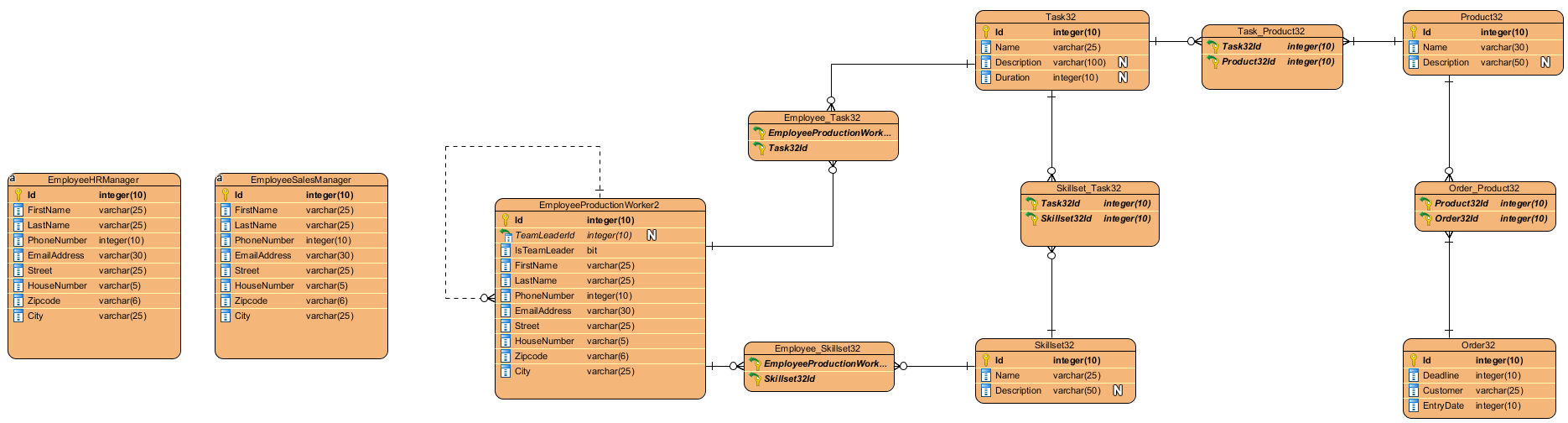
## DBO: Subtypering 2



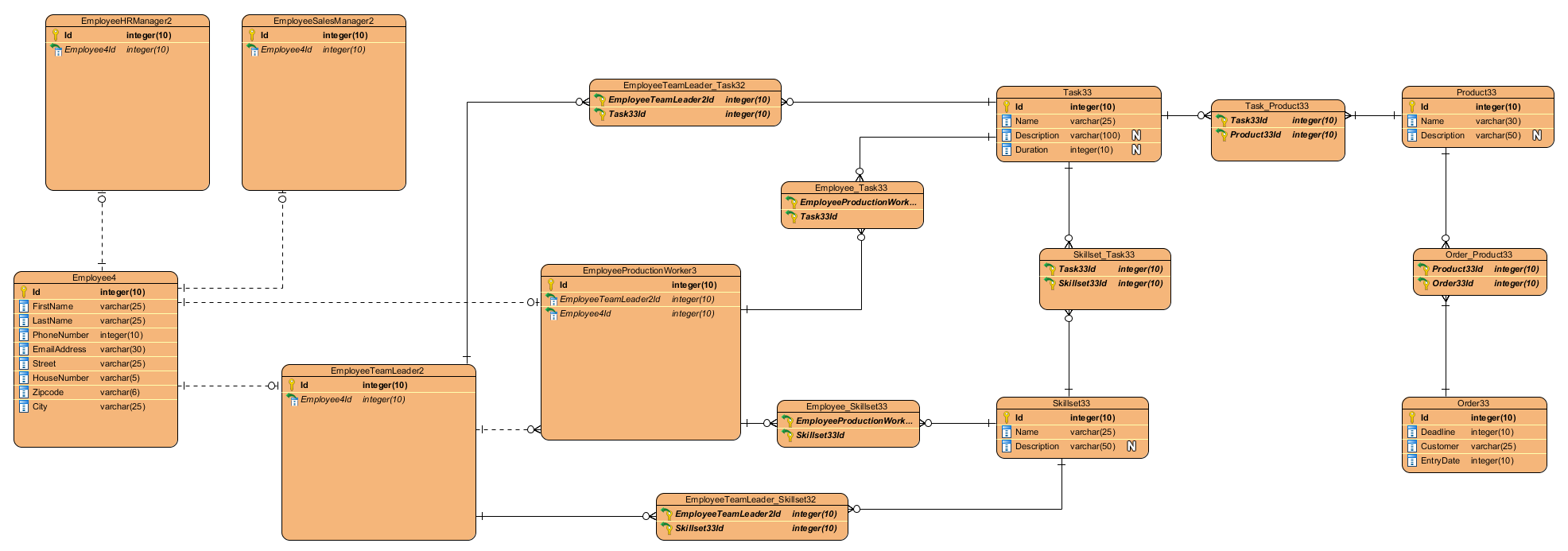
## DBO: Subtypering 3A



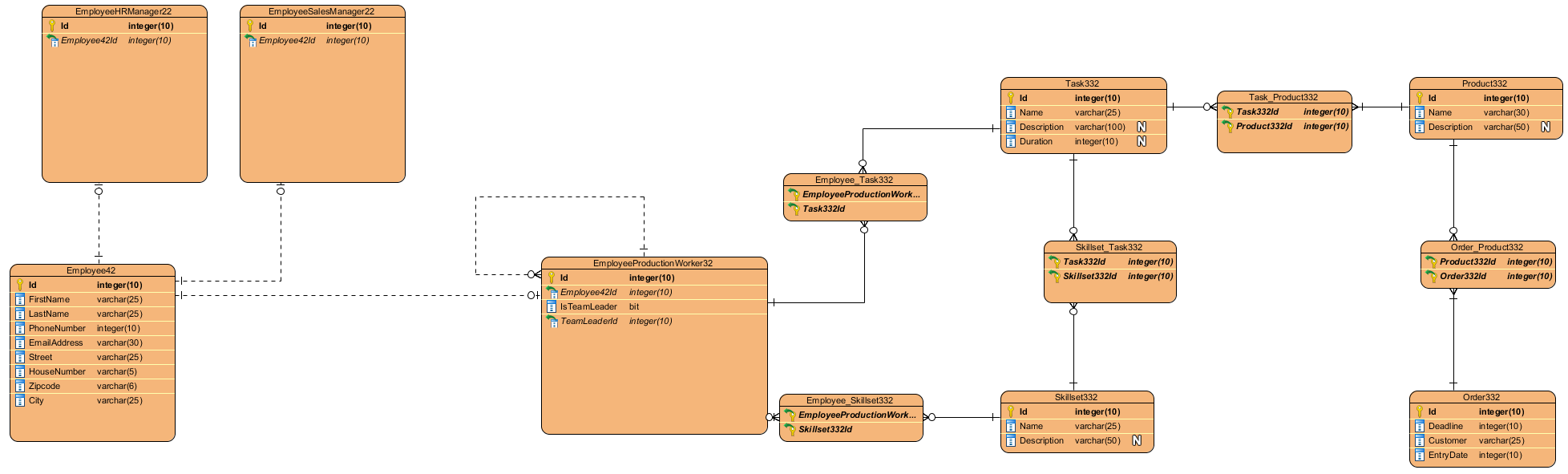
## DBO: Subtypering 3B



## DBO: Subtypering 4A



## DBO: Subtypering 4B



## Conclusie Versie 2

Na overleg met Alexander lijkt het een beter idee om voor de subtypering tussen Employee, HR Manager, Sales Manager en Production Worker subtypering 4 te gebruiken. Dit houdt in dat er een tabel is van Employee met generieke gegevens en 3 tabellen voor de subtypes van Employee. Hierdoor kan ik de verschillende relaties, die de ene subtypering wel heeft en de andere niet, afdwingen in het databaseontwerp. Anders zou ik allerlei triggers en constrains moeten schrijven om dit daarin af te dwingen.

Voor de subtypering tussen Production Worker en Team Leader gebruik ik subtypering 1. Dit houdt in dat alle teamleiders en productiemedewerkers in dezelfde tabel staan, maar een veld hebben waarmee wordt aangegeven welke type ze zijn. Volgens dit ontwerp zou een teamleider boven een teamleider kunnen staan, maar dit wil ik nog niet. Hiervoor zal ik dus een trigger of constrain moeten schrijven.

Ook heb ik nog een relatie vastgelegd tussen Order en EmployeeSalesManager.

## DBO: Na feedback met subtypering 4 en 1

