Inhoudsopgave

1 Data en informatie **1**
- 1.1 Doel . . . 1
- 1.2 Data en Informatie . . . 1
- 1.3 Data en IT toepassingen . . . 2
- 1.4 Databanken en DBMS . . . 3

2 Van vraag naar data **5**
- 2.1 Doel . . . 5
- 2.2 Analyse . . . 5
- 2.3 Datamodelleren . . . 6
  - 2.3.1 Conceptueel datamodel . . . 6
  - 2.3.2 Logisch datamodel . . . 7
  - 2.3.3 Fysiek datamodel . . . 8

3 Conceptueel datamodel **9**
- 3.1 Doel . . . 9
- 3.2 Entity Relationship Diagram . . . 10
- 3.3 Entiteiten en entiteittypes . . . 10
  - 3.3.1 Wat zijn entiteiten en entiteittypes? . . . 10
  - 3.3.2 Hoe herkennen we entiteittypes? . . . 11
  - 3.3.3 Notatie ERD . . . 12
  - 3.3.4 Hoe eraan beginnen? . . . 12
  - 3.3.5 Oefening 1 - Entiteittypes . . . 12
- 3.4 Attributen . . . 13
  - 3.4.1 Wat zijn attributen? . . . 13
  - 3.4.2 Hoe herkennen we attributen? . . . 13
  - 3.4.3 Notatie ERD . . . 14
  - 3.4.4 Oefening 1 - Attributen . . . 14
- 3.5 Relaties . . . 15
  - 3.5.1 Wat zijn relaties? . . . 15
  - 3.5.2 Hoe herkennen we relaties? . . . 15
  - 3.5.3 Notatie ERD . . . 16
  - 3.5.4 Redundantie . . . 16
  - 3.5.5 Graad . . . 17
  - 3.5.6 Kardinaliteit van een relatie . . . 20
  - 3.5.7 Oefening 2 . . . 25
  - 3.5.8 Oefening 1 - Relaties . . . 26
- 3.6 Soorten attributen . . . 26
  - 3.6.1 Sleutelattributen . . . 26
  - 3.6.2 Afgeleide attributen . . . 26

3.6.3 Samengestelde en enkelvoudige attributen . . . 27
3.6.4 Enkelwaardige en meerwaardige attributen . . . 28
3.6.5 Attribuut van een relatie . . . 28
3.6.6 Oefening 3 . . . 29
3.6.7 Oefening 1 - Verdergezet . . . 30
3.7 Zwakke en sterke entiteiten . . . 30
3.7.1 Oefening 4 . . . 31
3.7.2 Oefening 5 . . . 31
3.8 Extended Entity Relationship Diagram (EERD) . . . 31
3.8.1 Subtypes en supertypes . . . 31
3.8.2 Oefening 6 . . . 33

4 Logisch datamodel **35**
4.1 Doel . . . 35
4.2 Inleiding . . . 36
4.3 Databankmodellen . . . 37
4.3.1 Relationeel databankmodel . . . 37
4.3.2 Andere databankmodellen . . . 40
4.4 Van conceptueel naar logisch datamodel . . . 41
4.4.1 Stap 1: Entiteiten worden tabellen . . . 41
4.4.2 Stap 2: Elke tabel krijgt een primaire sleutel toegewezen . . . 41
4.4.3 Stap 3: Voor elke 1-1 en 1-N relaties zetten we de primaire sleutel over als vreemde sleutel . . . 42
4.4.4 Stap 4: Voor elke N-M relaties creëren we een tussentabel . . . 44
4.4.5 Stap 5: de speciale gevallen . . . 45
4.5 Datamodel normaliseren . . . 51

5 Fysiek datamodel **53**
5.1 Doel . . . 53
5.2 Inleiding . . . 53
5.3 Primaire sleutel . . . 53
5.4 Null of NOT NULL . . . 53
5.5 Gegevenstype . . . 53
5.6 Van logisch naar fysiek datamodel . . . 53

Oplossingen **55**

1

Data en informatie

1.1 Doel

In het volgende hoofdstuk willen we even stil staan over wat de concepten ‘data’ en ‘informatie’ exact betekenen en waarom dit belangrijk is voor jullie als (toekomstige) ontwikkelaars. Jullie moeten uit dit hoofdstuk het volgende onthouden:

- de betekenis van data en informatie
- de nood aan persistente data binnen IT toepassingen
- DBMS voor het efficiënt bouwen en onderhouden van databanken

Het vak Database Foundations moet jullie als student voorbereiden op het gebruiken van data in de IT toepassingen die jullie gaan creëren, configureren, onderhouden, ... Het moet jullie helpen om met data aan de slag te gaan in de IT oplossingen die jullie ontwikkelen. We willen in dit vak jullie ook bewust maken van een aantal concepten, en de uitdagingen die komen kijken bij het gebruik van data in IT oplossingen.

1.2 Data en Informatie

Voordat we jullie aan de slag laten gaan, willen we even stil staan met een aantal belangrijke begrippen. In eerste instantie willen we het even hebben over data, en waar het onderscheid gemaakt wordt met informatie. Beide termen worden vaak door elkaar gebruikt, maar er is wel degelijk een verschil.

Data verwijst naar specifieke feiten die de vorm kunnen aannemen van cijfers, getallen, woorden, karakters, ... Bijvoorbeeld ‘23’, ‘Turing’, ‘1912’, ‘Londen’, ‘Alan’ en ‘juni’ zijn elk op zich feiten die als data beschouwd kunnen worden.

Informatie verwijst naar de betekenis die gebruikers aan deze feiten gaan toekennen en de verbanden die gelegd wordt tussen deze feiten. Bijvoorbeeld, aan bovenstaande feiten kunnen de volgende informatie toekennen: ‘Op 23 juni 1912 werd Alan Turing in Londen geboren’. We combineren de data om er context en waarde aan toe te kennen.

We zijn gewend om data in te vullen in de verschillende papieren en digitale formulieren die we dagelijks invullen. We worden in de meeste gevallen niet om ons levensverhaal gevraagd, maar gericht om bepaalde stukken data, bijvoorbeeld naam, voornaam, geboortedatum, ... die in de structuur van een formulier gegoten zijn. Het proces is ontworpen om deze informatie bij te houden zodat op een later tijdstip deze terug beschikbaar en op basis van de data de nodige informatie kan gegeven worden.

Informatie wordt in het kader van IT oplossingen vaak in stukken gekapt tot atomaire data zodat deze makkelijk te structureren en te verwerken is. Dat komt ook vaak terug in de technologieën waar jullie mee aan de slag zullen gaan, dat het structureren van de data essentieel is voor efficiënte verwerking. In de interactie met de gebruiker zal de oplossing dan vaak zo ontworpen zijn dat de data terug tot informatie gevormd wordt om waarde te bieden aan de gebruikers. Bijvoorbeeld, bij jullie inschrijving aan de UCLL vullen jullie een formulier in waarbij de data die jullie invoeren op een gestructureerde manier worden opgeslagen. Wanneer jullie dan via de UCLL pagina de informatie van jullie inschrijving zouden bekijken, wordt dit op het scherm getoond. Het is de ontwikkelaar die zich ontfermt over het gestructureerd verzamelen van de data en die dan vanuit die structuur aan de gebruiker in de vorm van informatie aanbieden. De rol voor het ontwikkelen van de transformatie van informatie naar data en vervolgens terug naar informatie ligt bij jullie als (toekomstige) ontwikkelaar.

1.3 Data en IT toepassingen

IT toepassingen worden vaak ontwikkeld met het oog op het digitaliseren of automatiseren van manuele processen. Bijvoorbeeld, het online inschrijven van een student voor een opleiding aan de UCLL gebeurde vroeger op papier, maar is ondertussen gedigitaliseerd om het manuele werk en kans op fouten te verminderen, maar zeker ook om de verwerking van al die inschrijvingen (deels) te automatiseren. Data speelt hierin een belangrijke rol, door middel van digitale formulieren verzamelen we data en die data moet ook ergens voor een langere periode opgeslagen worden, willen we die later voor verwerking kunnen gebruiken en aan gebruikers kunnen tonen. Die opslag voor een langere periode, noemen we **persistente gegevensopslag**.

Bij het ontwerpen van een toepassing zal het team van ontwikkelaars een architectuur van de toepassing gaan bepalen. Hierbij wordt nagedacht welke componenten noodzakelijk zijn een goed werkend systeem te bouwen, bijvoorbeeld een ontwikkelde interface voor mobiele toestellen, communicatie met andere toepassingen, ... waarbij de verschillende componenten elke een bepaalde functie vervullen (dit komt later in jullie opleiding uitgebreid aan bod). In heel wat gevallen zal een vorm van persistente gegevensopslag onderdeel van de architectuur vormen. Door de verschillende decennia is er heel wat technologie ontwikkeld die persistente gegevensopslag mogelijk maakt. In het kader van dit vak richten we ons op databanken en DBMS aangezien dit de standaard oplossing is voor persistente gegevensopslag, maar zeker niet de enige.

1.4 Databanken en DBMS

Als ontwikkelaar zullen jullie in het kader van jullie toepassingen dus data verzamelen, verwerken en tonen aan gebruikers. Alle data die jullie verzamelen kan in een databank verzameld worden. Een **databank** wordt gezien als een verzameling van 'persistente data'. Met andere woorden, een databank omvat een set van data die voor een lange periode bewaard moeten worden. Bijvoorbeeld, het systeem van de studentenadministratie van de UCLL omvat de data betreffende studenten, inschrijvingen, opleidingsonderdelen, ... die samen in één databank omvat zitten.

Persistente data wordt opgeslagen in het permanente geheugen van een systeem. Het permanente geheugen of opslagmedium kan verschillende vormen aannemen, bijvoorbeeld harde schijven, magneetbanden, CD-ROM, ... De opslag van data komt met een bepaalde kost die sterk afhangt van het gebruikte opslagmedium. Vandaar is het zeer belangrijk voor een team van ontwikkelaars om stil te staan met de waarde van data die aan een databank wordt toegevoegd en een afweging te maken tussen welke data in de databank wordt opgenomen en welke niet. Dat gezegd zijnde, de kost van dataopslag is sinds de jaren 2000 zeer sterk gedaald, dus wat extra data bijhouden zal in heel wat gevallen geen problemen opleveren.

De technologie die we gebruiken om deze databanken op te bouwen, te onderhouden, te beheren, te beveiligen en te bevragen (data uit de databank halen) noemen we een **database management systeem** (DBMS). Er bestaan heel wat database management systemen en in het kader van dit vak maken we gebruik van PostgreSQL. Andere voorbeelden zijn MySQL, Oracle, Microsoft SQL Server, IBM DB2, MongoDB, Neo4j, ...

Database management systemen komen in heel wat verschillende vormen. Bij de keuze van een database management systeem wordt naar heel wat verschillende factoren gekeken, gaande van kostprijs van een licentie, de manier waarop de data wordt gestructureerd, data beveiliging, features, ... maar dit valt buiten het doel van dit vak.

2

Van vraag naar data

2.1 Doel

In het volgende hoofdstuk willen we even stil staan met hoe datasets en databanken tot stand komen. Jullie moeten uit dit hoofdstuk het volgende onthouden:

- datamodelleren als een stap in de analyse van IT toepassingen
- een stapsgewijze aanpak voor de ontwikkeling databanken
- conceptuele, logisch en fysiek datamodel

2.2 Analyse

In het hoofdstuk "Van data naar informatie"hebben we al kort toegelicht hoe we, gebruik makende van SQL, een bestaande dataset kunnen bevragen. In de volgende lessen gaan jullie hier nog verder op in om verder in te gaan op de mogelijkheden die SQL te bieden heeft. In parallel willen we ook een andere uitdaging toelichten, namelijk hoe komen datasets en databanken tot stand? Welk proces leidt tot goed gebouwde databanken?

Het proces voor het tot stand komen van een databank, is vergelijkbaar met dat van het bouwen van een huis. In eerste instantie moet er een vraag of behoefte zijn om iets te bouwen. Van zodra je weet dat je een huis gaat bouwen, neem je contact op met een architect. De architect gaat samen met de klant nadenken van wat er exact gebouwd moet worden. Daarbij gaat de architect, gebruik makende van zijn technische expertise, er over waken dat tijdens het ontwerpen van het huis een aantal basisprincipes gerespecteerd blijven. De architect tekent een grondplan dat met de opdrachtgever besproken wordt om zeker te zijn dat zijn interpretatie van de vraag overeenkomt met de verwachtingen van de klant. In latere fases zal de architect van dit plan meer gedetailleerde versies maken zodat aannemers, elektriciens, loodgieters, ... weten hoe de constructie gemaakt moet worden. Dit grondplan wordt goed bewaard zodat er bij eventuele verbouwingswerken een overzicht is waar bepaalde constructie-elementen (waterleidingen, elektriciteit, steunbalken, ...) zich bevinden.

Voor de bouw van databanken loopt het proces gelijkaardig. Een architect (die hebben we ook in IT) of een analist zal samen met gebruikers en ontwikkelaars nadenken welke data er in

een databank terecht moet komen, hoe het proces verloopt dat we trachten te automatiseren en wat de achterliggende bedrijfslogica is. Op basis van die gesprekken wordt er door de analist een datamodel uitgetekend dat besproken wordt met de relevante partijen. Dit model wordt dan in een aantal steppen verder in detail uitgewerkt tot een bouwplan voor een databank. Dit bouwplan laat de ontwikkelaars toe om enerzijds de databank te bouwen maar ook alvast van start gaan met de ontwikkeling van de toepassing die gebruik maakt van de databank. Want van zodra er afspraken zijn over de structuur van de databank, kan je De modellen vormen een belangrijke bron voor documentatie voor analisten en ontwikkelaars voor de verdere ontwikkeling en onderhoud van de systemen.

Het proces om een vraag naar een IT toepassing te vertalen naar een aantal technische vereisten noemen we in IT (behoefte-)analyse. Het deelproces waarbij we de vereisten voor data in kaart brengen, noemen we specifiek *datamodelleren*.

Analyse vormt een onderdeel van de ontwikkelcyclus van IT toepassingen. In latere vakken wordt dieper ingegaan op ontwikkelmethodologieën omdat er wel meerdere bestaan (o.a. agile, DevOps, waterval, rapid application development). Wat jullie op dit ogenblik moeten weten is dat datamodellering een onderdeel vormt van de analyse van IT toepassingen.

2.3 Datamodelleren

Datamodelleren verloopt in een aantal stappen. In elke stap proberen we een bepaalde vraag te beantwoorden met de relevante belanghebbenden en vervolgens een datamodel op te stellen al dan niet startend vanaf een eerder bestaand datamodel uit een vorige stap. Elke stap resulteert in een datamodel en bijbehorende documentatie.

Een *datamodel* is een visuele en schematische voorstelling van de data die we in een databank willen opnemen. Een datamodel bevat de verschillende elementen die samen de databank vormen en de relaties die er bestaan tussen deze elementen. We gebruiken hiervoor een gestandaardiseerde notatie die enerzijds het bouwen van een datamodel vereenvoudigt en anderzijds de leesbaarheid van een model makkelijk maakt. Als er een consensus bestaat wat we toevoegen aan een datamodel, hoe dit voorgesteld wordt en hoe dit geïnterpreteerd moet worden, kunnen de modellen makkelijk uitwisselbaar zijn tussen de verschillende belanghebbenden.

In ons proces onderscheiden we drie types van datamodellen: conceptueel, logisch en fysiek.

2.3.1 Conceptueel datamodel

De eerste stap is het uittekenen van een *conceptueel datamodel* waarbij we een antwoord definiëren op de vraag "Welke informatie is relevant voor de beoogde IT toepassing?". In dit datamodel creëren we een beeld op de *realiteit* van de gebruikers zonder hierbij technische details of technische beperkingen mee in rekening te nemen. De focus ligt daarentegen op de

relevante concepten en bedrijfslogica (business logic). Het conceptueel datamodel laat ook toe het bereik (scope) van het datamodel goed af te bakenen, zodat alle betrokken duidelijk zicht hebben op wat van belang is, en wat er niet meegenomen wordt in de toepassing.

Bij het uittekenen van het conceptueel model proberen we de volgende vragen te beantwoorden:

- Welke concepten zijn belangrijk voor onze applicatie?
- Hoe gaan we die concepten beschrijven?
- Welke verbanden liggen er tussen deze concepten?

Het conceptueel komt typisch tot stand als een samenwerking tussen analisten en gebruikers. De gebruikers beschrijven tijdens workshops, interviews, ... hoe ze vandaag (of in de toekomst) werken en welke informatie relevant is. De analist stelt bijkomende vragen om voldoende details te weten te komen en hoe al deze elementen verband houden. De analist verzamelt deze informatie en bundelt deze in een datamodel. Het datamodel dient als een communicatiemiddel tussen de verschillende betrokken partijen. Door het eenvoudige karakter van het datamodel kunnen gebruikers makkelijk evalueren of de interpretatie van de analisten correct is. Anderzijds bevat het voldoende informatie voor de analisten om hier verder mee aan de slag te gaan.

Heel belangrijk! Op dit niveau wordt er nog niet gesproken over technologie, enkel high-level welke informatie relevant is voor de gebruikers. Dat maakt het conceptueel datamodel ook zo waardevol, na deze stap kan het ontwikkelteam van de toepassing nog alle richtingen uit.

We gaan later dieper in op het Conceptueel datamodel en hoe dit tot stand komt.

2.3.2 Logisch datamodel

Van zodra we zicht hebben op de informatie die voor de gebruikers relevant is, proberen we de volgende vraag te beantwoorden: "Welk databankmodel is het meest geschikt voor de ontwikkeling van onze databank?". Er bestaan namelijk verschillende databankmodellen (niet te verwarren met datamodellen) waarbij elk van de databankmodellen een manier definiëren om data te structureren. Elk databankmodel heeft zijn voor- en nadelen of specifieke toepassingen, en een goede overweging is noodzakelijk.

De keuze voor een bepaald databankmodel wordt typisch genomen in samenspraak tussen de analisten (met zicht op de data die gemodelleerd wordt), de architect (met zicht op de brede technische omgeving waarin de toepassing wordt ontwikkeld) en de ontwikkelaars (met zich op de technologie die voor de toepassing gebruikt zal worden). De impact van de keuze is niet alleen relevant voor de databank, maar legt ook beperkingen op voor het DBMS en de technologie dat gebruikt wordt. Na de keuze voor een bepaald databankmodel vertaalt de analist het conceptuele datamodel naar een *logisch datamodel* rekening houdend met de specifieke regels van het databankmodel. Sommige databankmodellen gaan bijvoorbeeld een beperking leggen op het type van relaties dat er tussen de data kan bestaan. In dit geval gaan

we tijdens de omzetting de niet ondersteunde types van relaties omzetten naar types van relaties die wel ondersteund worden. We komen hier later uitgebreid op terug mocht dit op dit ogenblik wat abstract klinken.

In heel wat gevallen zal gekozen worden voor een *relationeel databankmodel*. In het geval van deze keuze wordt het logisch datamodel dan ook een relationeel datamodel genoemd. We leggen binnen dit vak de focus op het relationeel databankmodel en komen hier in een later hoofdstuk uitgebreid op in.

Het logisch datamodel is net omwille van de omzetting naar een specifiek databankmodel al een stukje technischer dan het conceptueel datamodel. Dit maakt de leesbaarheid van het datamodel ook wat moeilijker niet-IT medewerkers. Dat gezegd zijnde bevat het datamodel nog te weinig technische informatie voor de ontwikkelaars van de databanken, zodat een bijkomende stap nodig is, namelijk de omzetting voor een specifiek DBMS. Dus op naar het fysiek datamodel...

2.3.3 Fysiek datamodel

In de laatste stap ligt de focus op de vraag "Welk DBMS gaan we gebruiken om de data op te slagen?". In de ontwikkeling van het fysiek datamodel worden de beslissingen genomen hoe de data fysiek gestructureerd zal worden binnen de databank rekening houdend met het gekozen DBMS. Elk DBMS heeft zo zijn eigen mogelijkheden en uitdagingen waar een ontwikkelteam gebruik van kan maken of rekening mee moet houden. Daarnaast zijn er ook heel wat overwegingen die gemaakt moeten worden naar performantie, beveiliging, ... Al deze technische beslissingen komen samen in het fysiek datamodel.

In deze fase gaan architecten, database administrators en ontwikkelaars evalueren welk DBMS gebruikt zal worden. In heel wat gevallen ligt deze keuze al vast aangezien de meeste organisaties in hun architectuur al een keuze hebben gemaakt. Verschillende DBMS in dezelfde architectuur betekent bijkomende complexiteit en kosten, dus wordt in de meeste gevallen vermeden waar mogelijk.

Van zodra de keuze voor een DBMS vast ligt, gaan databast administrators en ontwikkelaars de structuur van de databank evalueren en technische details bepalen hoe de data concreet opgeslagen zal worden. Deze analyse resulteert in het fysiek datamodel waarin al deze informatie vervat zit. Het fysiek datamodel laat de ontwikkelaars toe om de nodige structureren te implementeren en de databank op te bouwen.

We komen in een later hoofdstuk meer in detail terug op het fysiek datamodel.

3

Conceptueel datamodel

3.1 Doel

In het volgende hoofdstuk willen we dieper in gaan op de verschillend bouwblokken van een conceptueel datamodel en het proces om een datamodel te ontwerpen. Hiervoor lichten we alvast een aantal belangrijke begrippen uit en gaan we ook de eerste stappen zetten om zelf eenvoudige conceptuele modellen te ontwerpen. We bekijken de volgende vragen:

- Wat zijn de bouwblokken van een conceptueel diagram? Wat is de rol van entiteiten, relaties en attributen als bouwblokken van een conceptueel datamodel?
- Welke stappen moeten genomen worden om een probleemstelling naar een conceptueel diagram te vertalen?

Jullie moeten de volgende termen kunnen uitleggen, de notatie in een ERD (waar relevant) en toelichten aan de hand van een voorbeeld:

- Conceptueel datamodel
- ERD diagram
- Entiteit en entiteittype (zwakke entiteittypes, sterke (of gewone) entiteittypes)
- Attribuut (sleutelattribuut, afgeleid attribuut, samengesteld attribuut, enkelvoudig attribuut, enkelwaardig attribuut, meerwaardig attribuut, attribuut van een relatie)
- Relatie (graad, kardinaliteit, 1-1 relatie, 1-N relatie, N-M relatie)
- Redundantie
- Subtypes, supertypes, specialisatie en generalisatie

3.2 Entity Relationship Diagram

Een **conceptueel datamodel** beantwoordt de vraag 'Welke informatie is relevant voor de beoogde IT toepassing?'. Dit datamodel geeft een beeld op de *realiteit* van de gebruikers en de informatie die hierin vervat zit, zonder hierbij technische details of technische beperkingen mee in rekening te nemen.

Een conceptueel datamodel is een schematische voorstelling van de relevante informatie. We streven naar een voorstelling die gemakkelijk op te stellen is en door een breed publiek gelezen kan worden. Om dat te bereiken, maken we gebruik van een gestandaardiseerde notatie, namelijk Entity Relationship Modelling. De diagrammen die gebruik maken van deze notatie noemen we **Entity Relationship Diagrams** (afgekort naar ERD).

ERD maakt gebruik van drie belangrijke concepten die de basis vormen van de modellen: entiteiten, relaties en attributen. We gaan elk van deze concepten uitgebreid toelichten en tonen hoe deze passen in het proces om tot een conceptueel datamodel te komen.

ERD notatie werd ontwikkeld in 1976 door Peter Chen en gaat dus al heel wat jaren mee. ERD werd later ook uitgebreid met enkele bijkomende concepten wat we dan het Extended Entity Relationship Model (EERD) noemen. De concepten die toegevoegd zijn, zijn specialisatie en generalisatie, en subklasses en superklasses. Deze concepten zullen we ook kort toelichten.

ERD is zeker niet de enige gestandaardiseerde notatie, zo heb je bijvoorbeeld ook UML. In de context van dit vak gebruiken we ERD notatie voor conceptuele en logische data modellen te beschrijven, omdat het een breed gebruikte standaard is. Heel wat mensen met een IT opleiding (en ook daarbuiten) kunnen vlot ERD lezen, wat het een uitstekende vorm van documenteren maakt.

3.3 Entiteiten en entiteittypes

3.3.1 Wat zijn entiteiten en entiteittypes?

De eerste belangrijke concepten binnen een ERD zijn **entiteiten** en **entiteittypes**. Een entiteit komt overeen met iets dat bestaat in de reële wereld. Dit kan zowel iets fysiek als abstract zijn zodat we kunnen spreken over objecten, personen, ... 'Sam Peters', 'Leuven', 'Een wagen met nummerplaat 1-AAA-001', ... zijn elk voorbeelden van entiteiten. Vanaf het ogenblik dat het bestaat en het relevant is, kunnen we het als een entiteit beschouwen.

Naast entiteiten hebben we ook entiteittypes. We definiëren entiteittypes als een verzameling van gelijkaardige entiteiten. Deze verzameling geven we vervolgens een naam geven die het type van entiteiten duidelijk omschrijft. Je zal merken dat we deze entiteittypes met een hoofdletter schrijven. Daar bestaat geen wet of regel voor, we doen dit voor duidelijkheid en consistentie.

Bijvoorbeeld, indien je een systeem ontwikkelt waarbij we informatie bijhouden over klanten, zouden 'Sam Peters' een klant zijn, en zou 'Klant' het entiteittype zijn dat de verzameling klanten beschrijft. We zeggen dat Sam een reële invulling is van een bepaald type. In het kader van dit opleidingsonderdeel gaan we een strikt onderscheid maken tussen enerzijds entiteit en entiteittype. We moeten echter toegeven dat dit in de beschikbare literatuur niet altijd het geval is. Houd dit in het achterhoofd wanneer je in boeken of online bepaalde dingen opzoekt.

3.3.2 Hoe herkennen we entiteittypes?

Om een ERD op te stellen, gaan we op zoek naar de relevante entiteittypes. Het herkennen van entiteittypes gebeurt typisch aan de hand van de zelfstandige naamwoorden die je tijdens een interview hoort of bij het doornemen van documenten leest of die gebruikt worden in de beschrijving van de oplossing, ... Woorden zoals 'klant', 'reservatie', 'product', 'winkel', 'aanvraag', 'wagen', ... zijn een goed startpunt. Vaak wil je dan informatie hierover bijhouden (zie attributen) wat van deze concepten goede kandidaten maakt.

Om de kandidaten te evalueren kunnen we de volgende vragen stellen:

- is het entiteittype iets wat **relevant** is voor onze toepassing? Anders gezegd, is het cruciaal dat we informatie over het entiteittype bijhouden? Heeft het een rol in de werking van de toepassing?

- is het entiteittype iets wat we willen en kunnen **beschrijven**? Hoe zouden we deze beschrijven? Welke informatie zouden we van een entiteittype kunnen bijhouden?

- kunnen we van het entiteittype makkelijk entiteiten bedenken? Hebben we voorbeelden van entiteiten?

- kunnen we meer dan één entiteit bedenken? Indien we maximaal één entiteit kunnen bedenken, dan is het vermoedelijk overbodig. Een voorbeeld van een overbodig entiteittype is bijvoorbeeld 'Hogeschool' in het geval van een administratiesysteem voor de UCLL Hogeschool. Aangezien we met slechts één hogeschool werken, zou dat het datamodel onnodig complex maken. De situatie verandert van zodra onze applicatie vanuit de associatie (combinatie van verschillende universiteiten en hogescholen) voor meerdere hogescholen ingezet zou worden, dan gaan we elk van die hogescholen informatie willen bijhouden.

Entiteiten nemen we niet op in een datamodel. Het is altijd wel handig om enkele voorbeelden van entiteiten mee op te nemen in jouw documentatie, want dit maakt het voor de personen die jouw ERD lezen makkelijker om zich voor te stellen wat je exact bedoelt.

3.3.3 Notatie ERD

Entiteittypes worden in een ERD weergegeven door een rechthoek maar daarin de naam van het entiteittype. Bijvoorbeeld de entiteittypes 'Klant', 'Product' en 'Wagen' worden als volgt weergegeven.

Figuur 3.1 Voorbeelden van de notatie van een entiteittype: 'Klant', 'Product' en 'Wagen'

3.3.4 Hoe eraan beginnen?

Zoals aangegeven kan je typisch entiteittypes herkennen als zelfstandige naamwoorden die gebruikt worden in een bepaalde tekst. Ons advies is om een bepaalde beschrijving te overlopen en alle zelfstandige naamwoorden aan te duiden. Dit zijn de kandidaten voor jouw entiteittypes. Voor elke van de kandidaten stel je de volgende vragen:

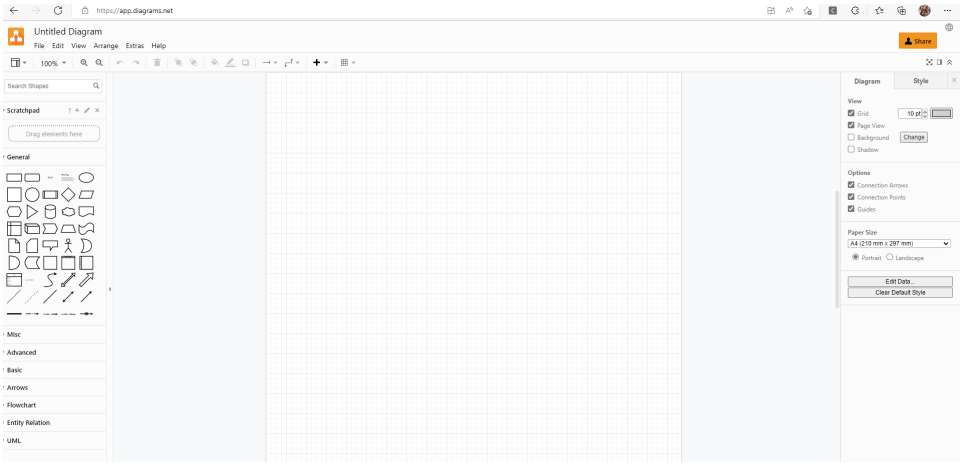
Jouw lijst gaat op dit ogenblik nog niet correct en volledig zijn, maar bij de volgende stappen, attributen en relaties, kan je jouw lijst van entiteiten nog verfijnen.

3.3.5 Oefening 1 - Entiteittypes

Oefening 1 van 'Conceptueel datamodel: oefeningen' geeft een beschrijving van een toepassing. Lees deze beschrijving en duid alles aan wat volgens jou een entiteittype zou kunnen zijn. Geen probleem om op dit ogenblik fouten te maken, in de volgende stappen gaan we dit verder verfijnen.

We raden jullie aan om diagrams.net te gebruiken om jullie diagrammen te tekenen. diagrams.net bestaat zowel in een online versie als een versie die je op jouw PC kan installeren. Jullie kunnen een YouTube video bekijken om wat meer te weten over hoe je ERD in diagrams.net maakt, houd wel in het achterhoofd dat bepaalde concepten voor jullie op dit ogenblik nog onbekend zijn.

Jullie mogen dit ook gerust op papier uitwerken, maar het kan soms handig zijn als je diagrammen kunt dupliceren of bepaalde componenten verslepen. Aan jullie de keuze.



Figuur 3.2 Screenshot diagrams.net

3.4 Attributen

3.4.1 Wat zijn attributen?

Entiteittypes beschrijven concepten die relevant zijn voor ons datamodel, maar bevatten eigenlijk geen data. Om aan te geven wat we specifiek over de verschillende entiteiten willen bijhouden, maken we gebruik van **attributen**. De attributen van een entiteittype beschrijven de gemeenschappelijke eigenschappen die we bijhouden voor elk van de entiteiten.

Voorbeelden van deze eigenschappen voor de entiteit ‘Sam Peters’ zijn ‘Sam’, ‘Peters’, ‘04/09/2001’, ‘21’, ‘Vilvoorde’ en ‘Bordspelen, rugby, reizen’ die respectievelijk waarden zijn voor de attributen ‘Voornaam’, ‘Naam’, ‘Geboortedatum’, ‘Leeftijd’, ‘Woonplaats’ en ‘Hobby’s’.

Een attribuut krijgt een algemene naam die de eigenschap beschrijft. De mogelijke waarden die de eigenschap kan aannemen noemen we het **domein** van een attribuut. Bijvoorbeeld, het attribuut met de naam ‘Woonplaats’ kan als domein ‘Vilvoorde’, ‘Leuven’, ‘Tienen’, ‘Landen’, ‘Heverlee’, ... hebben.

Er zijn verschillende types van attributen, maar daar komen we op een later moment op terug.

3.4.2 Hoe herkennen we attributen?

Attributen zijn eigenschappen van entiteittypes. Dus mocht je een kandidaat-entiteittype hebben, kan je op zoek gaan naar de verschillende elementen die iets zeggen over de entiteittypes en de data die we hierover willen bijhouden. Vaak geven constructies zoals ‘Over de klant willen we de volgende informatie bijhouden...’ een zeer duidelijke hint dat we data willen bijhouden over entiteittype ‘Klant’ met dan de nodige attributen. Helaas is het niet

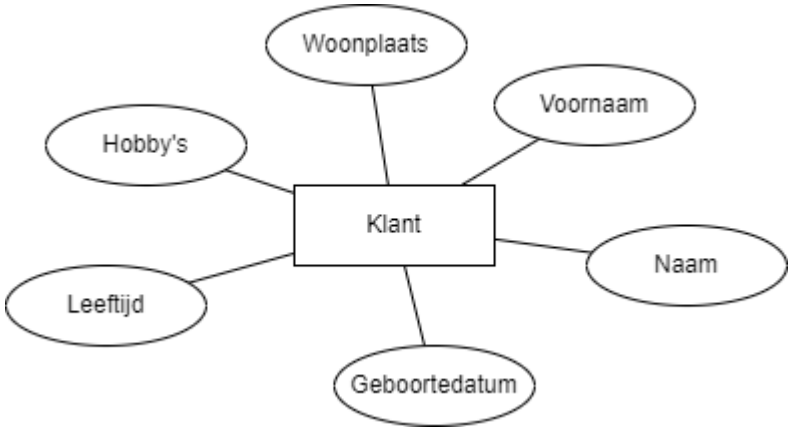
altijd even duidelijk aangegeven. Soms kan het makkelijker zijn om een bepaalde entiteit voor ogen te nemen. Wat willen we weten over onze klant ‘Sam Peters’?

Soms moet je voor bepaalde elementen de keuze maken of het een entiteittype is of een attribuut van een entiteittype. Enkele tips die we hierbij kunnen geven:

- Een entiteittype zonder één enkel attribuut is een lege doos. We hebben een concept waar we geen informatie over bijhouden. Wat is de relevantie van van dit entiteittype?
- Een entiteittype met slechts één attribuut, is mogelijk een attribuut van een ander entiteittype. Een uitzondering hierop bestaat indien dit entiteittype verschillende relaties heeft met andere entiteittypes, wat aan bod komt in het volgende onderdeel.

3.4.3 Notatie ERD

In de context van een ERD wordt een attribuut weergegeven door een ellips met hierin de naam van het attribuut. Deze ellips wordt vervolgens met een lijn verbonden met het entiteittype. Een attribuut kan niet gedeeld worden tussen verschillende entiteittypes!



Figuur 3.3 Voorbeeld van de notatie van attributen: het entiteittype ‘Klant’ met een aantal attributen ‘Voornaam’, ‘Naam’, ‘Geboortedatum’, ‘Leeftijd’, ‘Hobby’s’ en ‘Woonplaats’

Het domein van een attribuut, met andere woorden de mogelijke waarden die een attribuut kan aannemen, wordt niet weergegeven in een ERD.

3.4.4 Oefening 1 - Attributen

Ga terug naar oefening 1 van ‘Conceptueel datamodel: oefeningen’ en bepaald voor elke van de entiteittypes welke attributen je hiervan terugvindt in de tekst. De attributen moeten toelaten om de entiteiten te beschrijven. Stel jezelf de volgende vragen:

- welke eigenschappen moet ik voor elk van de entiteittypes bijhouden?

- welke informatie is noodzakelijk voor mijn applicatie? Kan ik deze informatie terugvinden in de attributen?
- ontbreken er bepaalde entiteittypes om de nodige data te structureren? Kan je bepaalde attributen moeilijk plaatsen bij de entiteittypes die je hebt geïdentificeerd?
- zijn er bepaalde entiteittypes die slechts één attribuut hebben? Zijn we zeker dat het hier gaat om een entiteittype met één attribuut, of een attribuut van een ander entiteittype? Bijvoorbeeld, willen we van een persoon een leeftijd bijhouden, dan zouden we een entiteittype 'Leeftijd' kunnen identificeren met als attribuut 'Leeftijd". Daarentegen kan dit ook gewoon een attribuut van het entiteittype 'Persoon' zijn, zonder dat we daarbij ons datamodel onnodig complex maken.

Breid jouw ERD in diagrams.net uit met jouw attributen. (TIP: je kan in diagrams.net een tab dupliceren. Je kan dan verder bouwen op het antwoord van een eerdere vraag, zonder dat je het antwoord op de eerste vraag kwijt speelt.)

3.5 Relaties

3.5.1 Wat zijn relaties?

Een entiteittype met attributen beschrijft concepten die we relevant vinden en de data die we hiervan willen bijhouden. **Informatie** wordt gecreëerd wanneer data gecombineerd wordt en verbanden gelegd worden. Om deze verbanden te beschrijven die bestaan tussen de verschillende concepten, maken we gebruik van relaties.

Een **relatie** verbindt een aantal entiteittypes en geeft het verband dat er bestaat tussen deze entiteittypes een naam. Daarbovenop laten relaties toe om het verband tussen de entiteittypes verder te nuanceren door het gebruik van een **maximum- en minimumkardinaliteit**. We bespreken kardinaliteit in 3.5.6.

Een voorbeeld van een relatie ontstaat wanneer Sam Peters in een supermarkt een flesje water koopt. Onze klanten zullen namelijk onze producten kopen. We hebben hierbij twee entiteittypes: 'Klant' (met als entiteit 'Sam Peters') en 'Product' (met als entiteit 'Flesje water'). Tussen beide entiteittypes creëren we een relatie 'Koopt'. Ons datamodel laat nu toe om informatie af te leiden van welke klanten welke producten kopen.

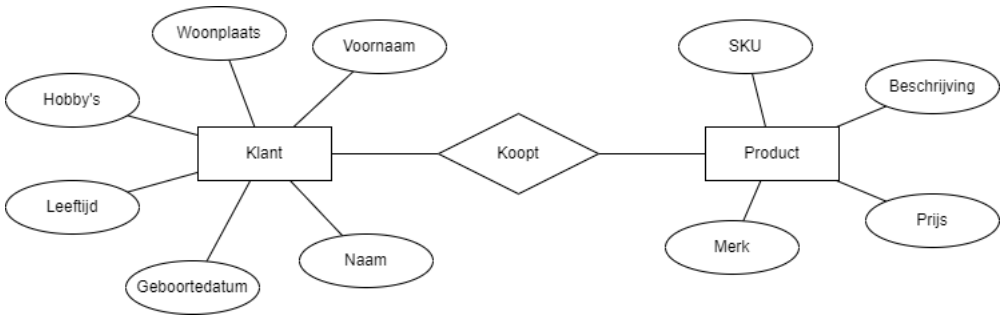
3.5.2 Hoe herkennen we relaties?

Relaties gaat om verbanden herkennen en benoemen. Vaak worden die verbanden beschreven in de vorm van 'is de <relatie> van' of een bepaald werkwoord. Je kan voor elke combinatie evalueren of er ergens een verband bestaat, maar opgelet dat je geen redundantie creëert. We bespreken redundantie in 3.5.4.

Een entiteittype dat met geen enkele ander entiteittype verbonden is, is meestal een teken dat er een relatie ontbreekt, of dat het geïsoleerde entiteittype niet relevant is.

3.5.3 Notatie ERD

Een relatie wordt in een ERD getekend als een ruit waarin we met een naam de relatie benoemen. De ruit wordt verbonden met de relevante entiteittypes. Voor elk van de entiteittypes geven we de kardinaliteit aan.



Figuur 3.4 Een voorbeeld van de notatoe van een relatie: een relatie 'Koopt' tussen twee entiteittypes 'Klant' en 'Product'. Elk entiteittype heeft een aantal attributen.

3.5.4 Redundantie

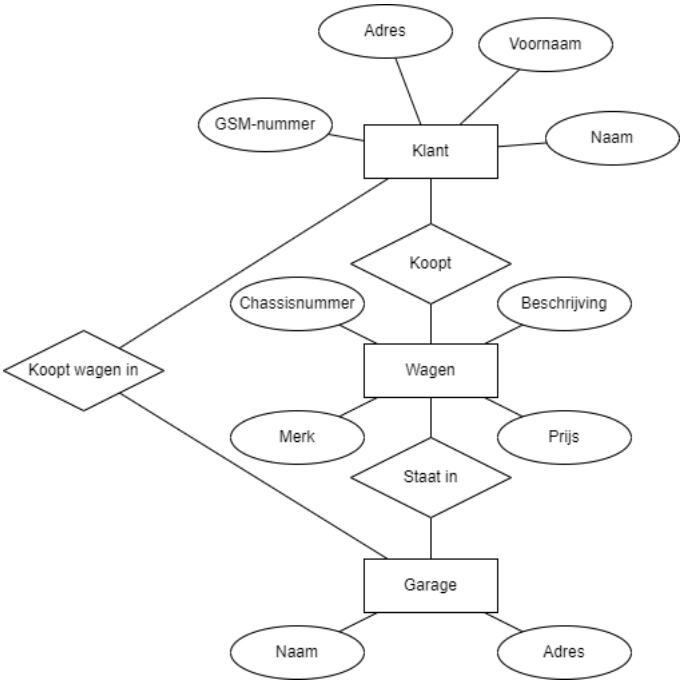
De bedoeling van een ERD is om een zo eenvoudig mogelijke weergave te hebben van de nodige informatie. We willen vermijden dat we meerdere keren hetzelfde zeggen. Dit noemen we **redundantie**.

Redundantie ontstaat hoofdzakelijk wanneer we relaties toevoegen die afgeleid kunnen worden uit andere relaties. Bijvoorbeeld, een klant koopt een wagen in één van de garages. We weten welke klant welke wagen koopt. We weten ook in welke garage een wagen verkocht wordt. Indien we willen weten in welke garage een klant een wagen koopt, kunnen we de relatie 'Koopt wagen in' tussen de entiteittypes 'Klant' en 'Garage' toevoegen.

Maar deze relatie is redundant. We kunnen deze informatie altijd afleiden uit de andere relaties. De relatie 'Koopt wagen in' nemen we niet mee op in het ERD.

Zeggen dat we redundantie kunnen afleiden uit elke lus in ons ERD, is kort door de bocht. Het klopt dat redundantie vaak de vorm van een cyclus aanneemt. Maar je moet wel altijd rekening houden met wat je specifiek wil zeggen en de kardinaliteit van de relaties.

Stel bijvoorbeeld dat we willen bijhouden dat een klant een bepaalde garage bezoekt, dan kunnen we een relatie 'Bezoekt' definiëren tussen de entiteittypes 'Klant' en 'Garage'. Het bezoek leidt mogelijk niet tot een aankoop, dus dat willen we wel apart bijhouden.



Figuur 3.5 Een redundante relatie 'Koopt wagen in' die afgeleid kan worden uit de combinatie van relaties 'Koopt' en 'Staat in' die respectievelijk de entiteittypes 'Klant' en 'Wagen' en 'Wagen' en 'Garage' met elkaar verbinden.

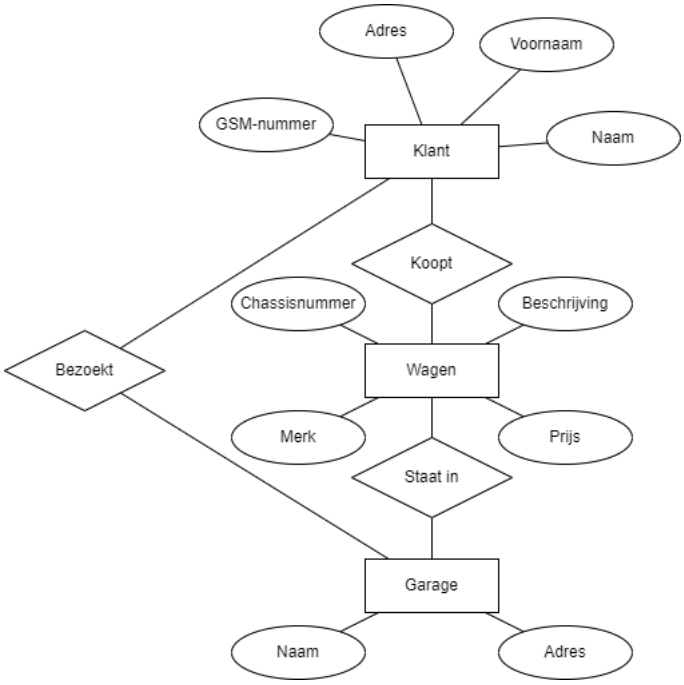
Stel bijvoorbeeld dat een wagen bij verschillende garages kan staan, dan kunnen we ook (uit bovenstaande ERD) niet afleiden bij welke garage de wagen gekocht werd. Dit kan je oplossen met een 'Attribuut van een relatie'. We bespreken dit in 3.6.5.

3.5.5 Graad

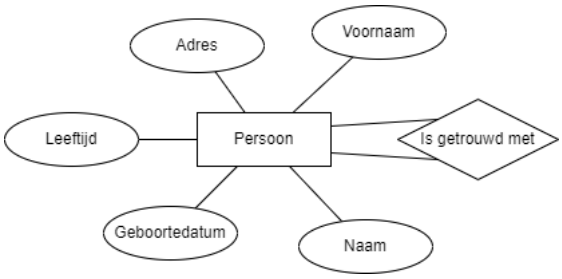
In de meeste gevallen zal een relatie *twee* entiteittypes met elkaar verbinden, maar dat hoeft niet per se het geval te zijn. Het is ook mogelijk dat een entiteittype *naar zichzelf* verwijst, of dat er een relatie bestaat tussen *drie of meer* entiteittypes. Het aantal entiteittypes dat een relatie met elkaar verbindt, noemen we de **graad** van een relatie.

In geval van 1 entiteittype, spreken we over een **unaire relatie** of een relatie van de eerste graad. In geval van 2 entiteittypes, spreken we over een **binaire relatie** of een relatie van de tweede graad. In geval van 3 entiteittypes, spreken we over een **ternaire relatie** of een relatie van de derde graad. Enzovoort. De meeste relaties zullen binair zijn, maar unair, ternair, ... komen zeker voor.

Een voorbeeld van een *unaire relatie* is 'Is getrouwd met' waarbij een entiteit van het entiteittype 'Persoon' getrouwd is met een entiteit van hetzelfde entiteittype.



Figuur 3.6 Een voorbeeld van een niet-redundante relatie 'Bezoekt' tussen de entiteittypes 'Klant' en 'Garage' die bijkomende informatie aangeeft.

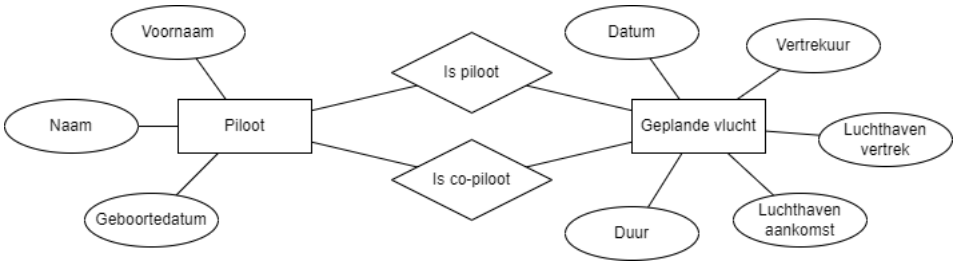


Figuur 3.7 Voorbeeld van een unaire relatie 'Is getrouwd met' die het entiteittypes 'Persoon' met zichzelf verbindt.

We gaan er voor dit voorbeeld vanuit dat de twee entiteiten niet dezelfde zijn. Maar in principe kan het ook gaan om een relatie dat een entiteit met zichzelf heeft. Een relatie waarbij we aangeven dat een minister opgevolgd wordt door een andere minister zouden we kunnen beschrijven door de relatie 'Volgt op' die twee entiteiten van het entiteittype 'Minister' met elkaar verbindt. Het kan gebeuren dat een minister zichzelf opvolgt.

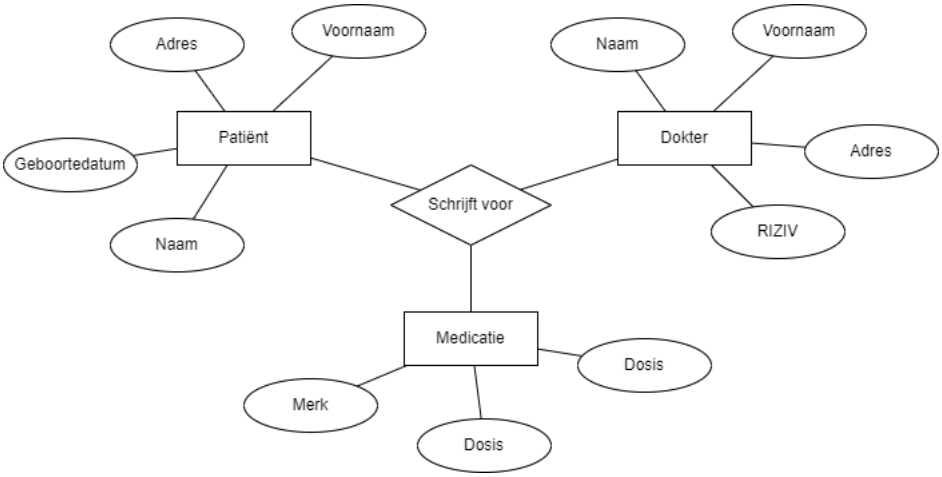
Een voorbeeld van een *binaire relatie* is 'Is piloot' waarbij een entiteit van het entiteittype 'Piloot' wordt toegekend aan een entiteit van het entiteittype 'Geplande vlucht'. Zoals onderstaande voorbeeld ook toont, is het mogelijk om meerdere relaties te hebben tussen twee entiteittypes aangezien we ook een binaire relatie 'Is co-piloot' hebben gedefinieerd. Voor een geplande vlucht zal een piloot als (hoofd-)piloot worden toegekend en een andere piloot

als co-piloot. Op basis van het model houden we de informatie van beide rollen bij.



Figuur 3.8 Voorbeeld van twee binaire relaties: de relatie 'Is piloot' en de relatie 'Is co-piloot' die het entiteittypes 'Piloot' en 'Geplande vlucht' verbinden.

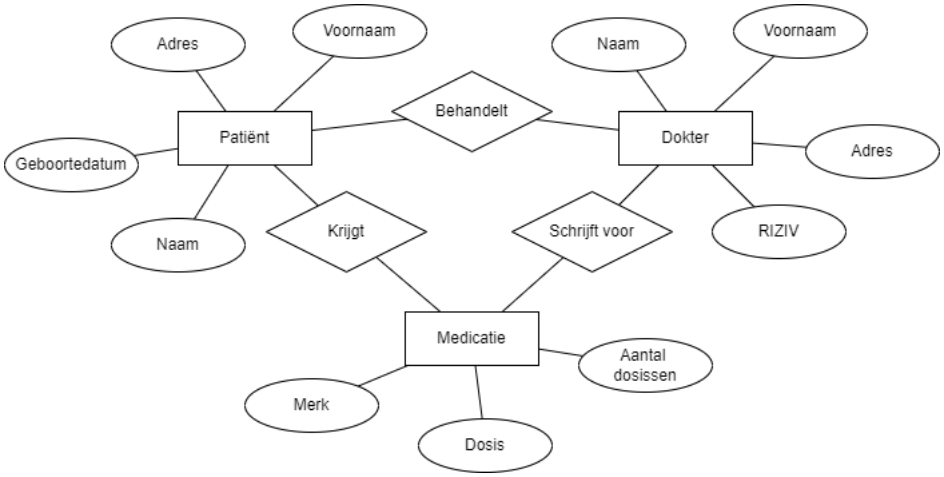
Een voorbeeld van een *ternaire relatie* is 'Schrijft voor' waarbij telkens entiteiten van entiteittype 'Patiënt', 'Dokter' en 'Medicatie' met elkaar verbonden worden.



Figuur 3.9 Voorbeeld van een ternaire relatie 'Schrijft voor' die de entiteittypes 'Patiënt', 'Dokter' en 'Medicatie' verbindt.

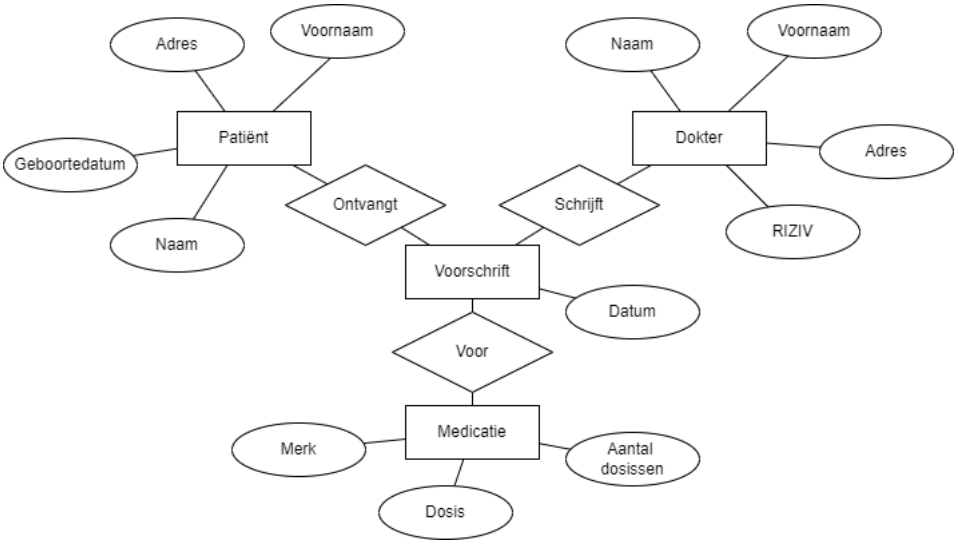
Voor het verband dat er tussen de drie bestaat, is het essentieel dat alle drie entiteittypes betrokken zijn. Mochten we dit vervangen door binaire relaties, zouden we namelijk informatie verliezen. Onderstaande laat ons wel toe om te achterhalen welke patiënt welke medicatie krijgt, maar niet door welke dokter het voorgeschreven wordt. Het laat ons toe om te achterhalen welke patiënt door welke dokter behandelt wordt, maar niet welke medicatie hierbij voorgeschreven wordt. En ten slotte weten we ook welke dokter welke medicatie voorschrijft, maar niet voor welke patiënt. Kortom elk van drie entiteittype geven een stukje informatie.

Alternatief kan bovenstaande ternaire relatie gemodelleerd worden als een entiteittype 'Voorschrift'. Hierbij krijgt de unieke combinatie van de drie entiteiten van de entiteittypes 'Patient', 'Dokter' en 'Medicatie' een entiteit van entiteittype 'Voorschrift'. Het attribuut 'Datum'



Figuur 3.10 Een voorbeeld hoe drie binaire relaties die geen goed alternatief vormen voor de ternaire relatie.

laat bijkomend ook toe om bijkomende informatie toe te voegen, bijvoorbeeld op welke datum het voorschrift werd geschreven.



Figuur 3.11 Een nieuw entiteittype 'Voorschrift' dat een de entiteittypes 'Patient', 'Dokter' en 'Medicatie' verbindt als alternatief op de ternaire relatie 'Schrijft voor'

3.5.6 Kardinaliteit van een relatie

Kardinaliteit beschrijft *hoeveel* relaties van een bepaald type een entiteit moet en kan hebben. Elke kardinaliteit bestaat uit twee delen, een minimum- en een maximumkardinaliteit. Dit wordt voor elk entiteittype binnen de relatie individueel bepaald.

Het doel van kardinaliteit is om meer informatie mee te geven over de aard van de relatie. Deze informatie is van belang wanneer we ons datamodel omzetten naar een logisch en fysiek datamodel. Een foute definitie van kardinaliteit leidt tot problemen en fouten in het gebruik van de data.

Binnen een ERD bestaan er verschillende manieren om kardinaliteit te noteren. In het kader van dit vak maken we gebruik van de (min,max)-notatie. Naast de (min,max)-notatie gaan we bij de logische datamodellen gebruik maken van de kraaienpootnotatie, zodat jullie beide kennen.

Minimumkardinaliteit (of participatierestrictie)

De **minimumkardinaliteit** geeft aan of een relatie *optioneel of verplicht* is voor alle entiteiten van een specifiek entiteittype. Er zijn twee mogelijke waarden: '0' en '1'. De waarde '0' geeft aan dat de relatie voor het specifiek entiteittype optioneel is. Optioneel betekent dat een entiteit kan bestaan zonder dat een relatie bestaat met andere entiteiten van de entiteittypes in de relatie. De waarde '1' geeft aan dat die verplichting wel bestaat. Met andere woorden, een entiteit van dit entiteittype kan enkel bestaan indien deze minstens 1 relatie heeft met entiteiten van de andere entiteittypes binnen de relatie.

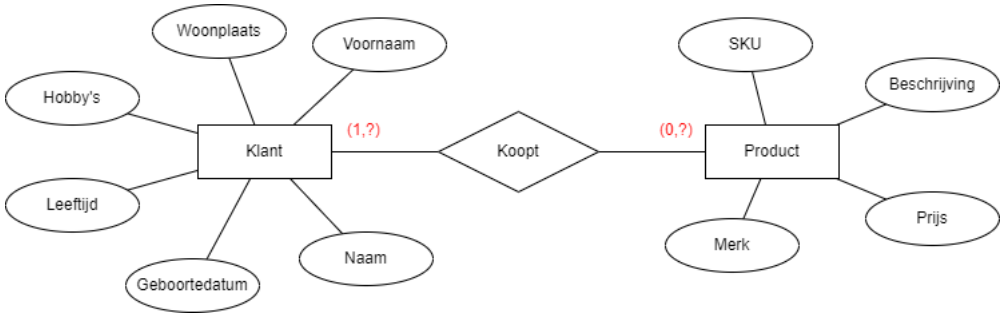
Dit kan mogelijk wat abstract klinken, dus laten we dit even aan de hand van voorbeelden toelichten. Een eerste voorbeeld is de relatie 'Koopt' tussen de entiteittypes 'Klant' en 'Product'. Stel nu de volgende vraag: kan een product bestaan zonder dat dit gekocht wordt door een klant? Het antwoord is 'ja', want nieuwe producten die we nog niet verkocht hebben, kunnen wel degelijk aanwezig zijn in het systeem zonder dat een klant die al gekocht heeft. De minimumkardinaliteit van het entiteittype 'Product' in de relatie 'Koopt' is dus '0'.

De volgende vraag die we stellen: kan een klant bestaan zonder dat deze een product heeft gekocht. Dat is al moeilijker eenduidig te beantwoorden, want dit gaat erg afhangen van hoe er in de winkel wordt gewerkt. Indien een klant zich vooraf kan registreren zonder dat hij een aankoop doet, kan een klant bestaan zonder dat hij een product heeft gekocht, en is de minimumkardinaliteit '0'. Maar mocht een klant pas aangemaakt worden van zodra deze een product koopt, dan is de waarde van de minimumkardinaliteit '1'.

We gaan even er van uit dat een klant pas klant wordt van zodra deze iets koopt, dus met een minimumkardinaliteit van '1'.

Binnen een ERD gaan we de minimumkardinaliteit aanduiden waar de lijn komende van de ruit die de relatie voorstelt het vierkant dat het entiteittype voorstelt, raakt. We schrijven (0,...) of (1,...).

In het kader van de minimumkardinaliteit is de waarde '1' de meest beperkende, omdat we eisen dat er minstens één relatie moet zijn. Bij twijfel geven we het advies telkens voor '0' te gaan of beter nog, geen veronderstelling te maken en gericht de vraag te stellen of de relatie al dan niet optioneel is voor het entiteittype.



Figuur 3.12 Een voorbeeld van de minimumkardinaliteit in (min,max) notatie voor elk van de entiteittypes in de relatie 'Koopt'.

Maximumkardinaliteit (of kardinaliteitsrestrictie)

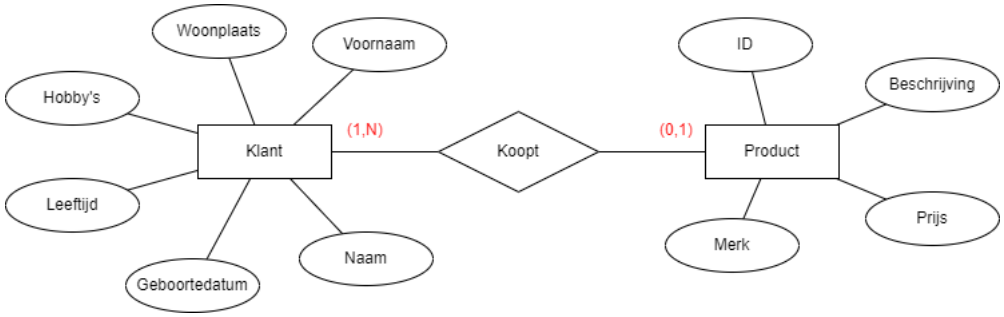
De maximumkardinaliteit geeft aan of een entiteit van een entiteittype aan meerdere relaties van hetzelfde type kan deelnemen. We hebben twee mogelijke waarden: 1 of N. De waarde 1 geeft aan dat een entiteit van een entiteittype maximaal 1 relatie van een bepaald type kan hebben. De waarde N geeft aan dat een entiteit van een entiteittype meerdere relaties van een bepaald type kan hebben.

Laten we dit opnieuw even toelichten aan de hand van ons eerder voorbeeld waarbij klanten producten kopen. We stellen onszelf de volgende vraag: kan een klant meerdere producten kopen? Het antwoord is 'ja', wat betekent dat we een maximumkardinaliteit hebben van 'N'. Als we dit combineren met de minimumkardinaliteit, dan kunnen we stellen dat een klant minimaal '1' en maximaal 'N' producten kan kopen, samengevat als (1,N);

De volgende vraag die we stellen, kan een product door meerdere klanten worden gekocht? Dit is opnieuw moeilijk eenduidig vast te leggen en zal afhangen van het type product we verkopen. Indien we spreken over bijvoorbeeld een supermarkt, dan kunnen we een flesje water beschouwen als een product, en kunnen meerdere personen een flesje water kopen. We houden geen overzicht bij welke klant een specifiek flesje water heeft gekocht. In dat geval is de maximumkardinaliteit 'N'. Maar indien het gaat om iets exclusiefs (bijvoorbeeld een wagen, uniek kunstwerk, een laptop, ...) dan kan dat uniek stuk maar aan 1 klant verkocht worden. En dan is de maximumkardinaliteit dus '1'.

Voor dit voorbeeld gaan we even er van uit dat een product maar aan 1 klant verkocht kan worden, dus met een maximumkardinaliteit van '1'.

Binnen een ERD gaan we de maximumkardinaliteit achter de minimumkardinaliteit toevoegen. We schrijven (<minimumkardinaliteit>,1) of (<minimumkardinaliteit>,N). Wanneer binnen eenzelfde type relatie meerdere keren een maximumkardinaliteit van N voorkomt, gaan we differentiëren door in plaats van N ook M, O, P, ... te gebruiken. Daarbij willen we benadrukken dat deze telkens andere waarden voorstellen, maar telkens wel groter dan '1'.



Figuur 3.13 Een voorbeeld van de minimum- en maximumkardinaliteit in (min,max) notatie voor elk van de entiteittypes in de relatie 'Koopt'.

Hoe moet je bovenstaande figuur (Figuur 3.13) lezen? 'Product' heeft in de relatie 'Koopt' een minimumkardinaliteit van '0', wat betekent dat een product kan bestaan zonder verkocht te zijn geweest. De maximumkardinaliteit van 'Product' in de relatie 'Koopt' is '1', wat betekent dat een product maar door één klant gekocht kan worden. 'Klant' heeft in de relatie 'Koopt' heeft een minimumkardinaliteit van '1', wat betekent dat een klant niet kan bestaan zonder dat deze een product heeft gekocht en er een relatie bestaat met minstens 1 product. De maximumkardinaliteit van 'Klant' in de relatie 'Koopt' is 'N', wat betekent dat een klant meerdere producten kan kopen en dus een relatie kan hebben met meerdere producten.

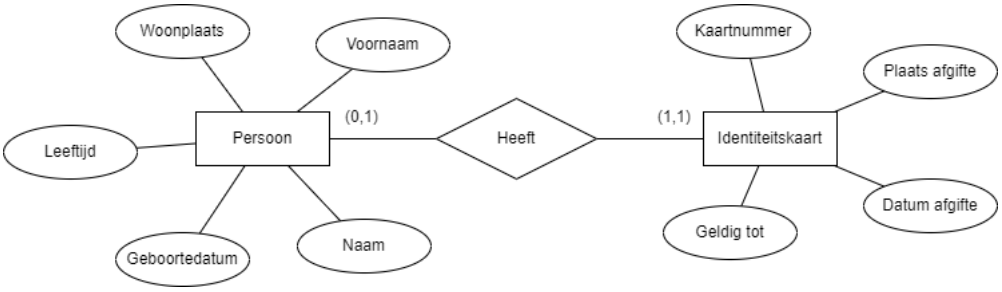
Types relaties: 1-1 relatie

Onder de binaire relaties, onderscheiden we 3 verschillende types. Het eerst type is een 1-1 relatie (één-op-één-relatie) waarbij de maximumkardinaliteit van beide entiteittypes binnen de relatie '1' is. Elke entiteit van het ene entiteittype kan dus maximaal 1 relatie hebben met een entiteit van het andere entiteittype. En dit geldt in beide richtingen.

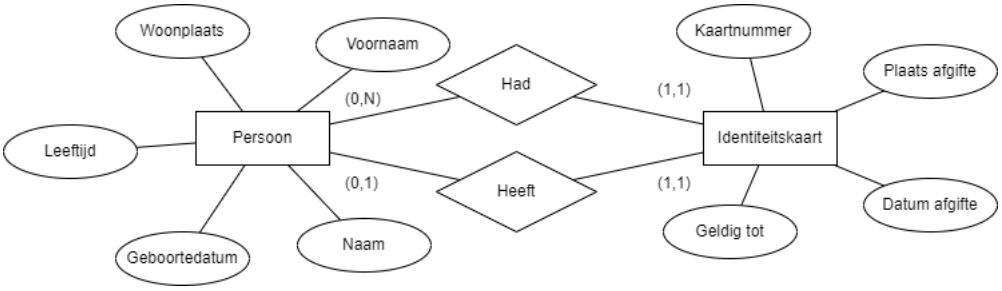
Een voorbeeld van een 1-1 relatie is een relatie 'Heeft' waarbij een entiteit (bijvoorbeeld 'Sam Peters') van het entiteittype 'Persoon' maximaal 1 entiteit van het entiteittype 'Identiteitskaart' kan hebben, en elke entiteit van het entiteittype 'Identiteitskaart' kan maximaal aan één entiteit van het entiteittype 'Persoon' toebehoren. Elke persoon kan maximaal één identiteitskaart hebben, en elke identiteitskaart behoort tot maximaal één persoon.

In bovenstaande voorbeeld gaan we ervan uit dat historiek niet bijgehouden wordt, maar een persoon kan doorheen de jaren wel meerdere identiteitskaarten hebben. Maar dat kunnen we oplossen door twee verschillende relaties te definiëren: 'Heeft' en 'Had'. Een persoon kan maximaal 1 identiteitskaart hebben, maar doorheen de tijd kan hij er wel meerdere gehad hebben.

De minimumkardinaliteit speelt hier geen rol in deze typering. Er wordt enkel naar maximumkardinaliteit gekeken.



Figuur 3.14 Een voorbeeld van een 1-1 relatie.



Figuur 3.15 Een voorbeeld van een 1-1 relatie.

Types relaties: 1-N relatie

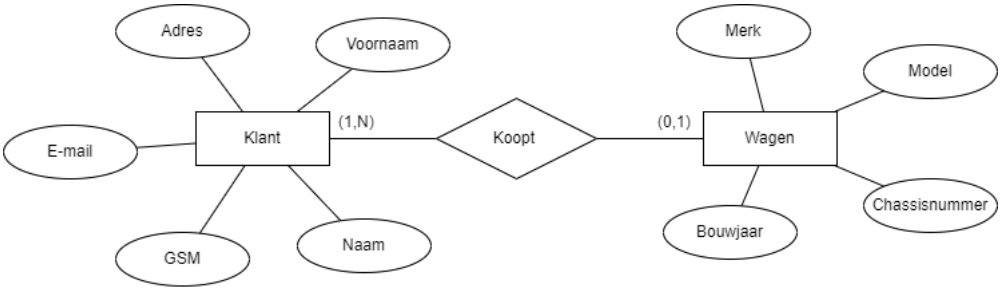
Een tweede type is een 1-N relatie (één-op-meer-relatie of één-op-veel-relatie) waarbij de maximumkardinaliteit van het ene entiteittype 'N' is en deze van het andere entiteittype '1' is. Elke entiteit van het entiteittype met maximumkardinaliteit 'N' kan een relatie hebben met meerdere entiteiten van het entiteittype met maximumkardinaliteit '1'. Omgekeerd kan elke entiteit van het entiteittype met maximumkardinaliteit '1' maximaal met 1 entiteit van het entiteittype met maximumkardinaliteit 'N' een relatie hebben.

Een voorbeeld van een 1-N relatie is een relatie 'Koopt' waarbij een entiteit (bijvoorbeeld 'Sam Peters') van het entiteittype 'Klant' meerdere entiteiten van het entiteittype 'Wagen' kan kopen, maar elke entiteit van het entiteittype 'Wagen' kan maximaal aan één entiteit van het entiteittype 'Klant' worden verkocht. Een klant kan meerdere wagens kopen, maar elke wagen kunnen we maximaal één keer verkopen (we houden tweedehandsverkoop even buiten beschouwing). Belangrijk is dit voorbeeld is dat elke wagen individueel identificeerbaar is en dat het relevant is om te weten welke klant welke wagen heeft gekocht.

Ook bij deze typering speelt minimumkardinaliteit geen rol.

Types relaties: N-M relatie

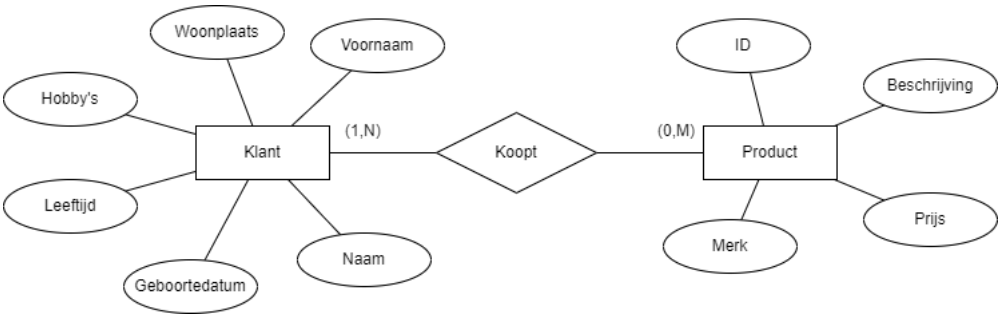
Een derde type is een N-M relatie (meer-op-meer-relatie of veel-op-veel-relatie) waarbij de maximumkardinaliteit van beide entiteittypes 'N' is. Elke entiteit van het ene entiteittype kan



Figuur 3.16 Een voorbeeld van een 1-N relatie.

dus meerdere relaties hebben met een entiteit van het andere entiteittype. En dit geldt in beide richtingen.

Een voorbeeld van een N-M relatie is een relatie 'Koopt' waarbij een entiteit (bijvoorbeeld 'Sam Peters') van het entiteittype 'Klant' meerdere entiteiten van het entiteittype 'Product' kan kopen, en elke entiteit van het entiteittype 'Product' kan door meerdere entiteiten van het entiteittype 'Klant' worden gekocht. Een klant kan meerdere producten kopen, elk product kan door meerdere klanten worden gekocht. Belangrijk is dit voorbeeld is dat het niet relevant is om elke product individueel te identificeren. We willen wel bijhouden dat een klant 'Sam Peters' een flesje water heeft gekocht, maar niet welk exact flesje dat dan is, in tegenstelling tot het voorbeeld met de wagen, waar we dit wel willen bijhouden.



Figuur 3.17 Een voorbeeld van een N-M relatie.

3.5.7 Oefening 2

Ga naar oefening 2 van 'Conceptueel datamodel: oefeningen' en lees elk van de relaties die beschreven wordt. Zet deze relaties om naar een ERD in diagrams.net, vermeld ook telkens de correcte kardinaliteiten en het type relatie.

3.5.8 Oefening 1 - Relaties

Ga terug naar oefening 1 van 'Conceptueel datamodel: oefeningen' en ga op zoek naar de verschillende relaties die er bestaan tussen de entiteittypes. Beschrijft de relaties met een duidelijke naam en geef de kardinaliteit aan voor elk van de entiteittypes van de relatie. Stel hierbij de volgende vragen:

- Laten de relaties toe om alle verbanden en informatie te achterhalen?
- Zijn alle entiteittypes met elkaar verbonden?

Breid jouw ERD uit met jouw relaties.

3.6 Soorten attributen

3.6.1 Sleutelattributen

Om entiteiten uniek te identificeren, maken we gebruik van **sleutelattributen**. Sleutelattributen van een entiteittype zijn attributen die toelaten om elke entiteit van een entiteittype uniek te benoemen.

Een voorbeeld van een sleutelattribuut voor het entiteittype 'Student' is het attribuut 'Studentnummer'. Elke student is uniek identificeerbaar aan de hand van zijn of haar studentnummer. Voor een wagen heb je de keuze, je kan bijvoorbeeld chassisnummer of nummerplaat gebruiken. Wat onderzoek zou je leren dat chassisnummer een betere keuze is, dus gebruik je dat als sleutelattribuut.

In een ERD gaan we sleutelattributen aanduiden door de naam van de attributen te onderlijnen. Indien we meerdere attributen nodig hebben om een entiteit te identificeren, onderlijnen we elk van de nodige attributen. Als we de keuze hebben tussen verschillende attributen, onderlijnen we slechts één van de opties.

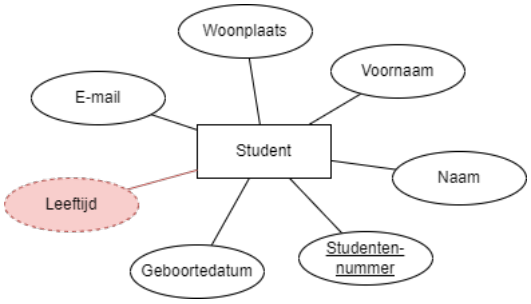
Sleutels vormen een belangrijke componenten van het relationeel databankmodel, dus hier komen we nog uitgebreid op terug in het kader van het logisch datamodel.

3.6.2 Afgeleide attributen

Attributen kunnen in sommige gevallen afgeleid worden op basis van andere attributen. We spreken in dat geval over **afgeleide attributen**.

Een voorbeeld van een afgeleid attribuut is het attribuut 'Leeftijd' wanneer we ook het attribuut 'Geboortedatum' ter beschikking hebben. Met behulp van een berekening op basis van deze datum kunnen we telkens ook de leeftijd bepalen.

Een afgeleid attribuut kan wordt in een ERD aangeduid door middel van een ellips die met een stippellijn getekend is waarin de naam van het attribuut wordt geplaatst.



Figuur 3.18 Een voorbeeld van een entiteittype 'Student' met een afgeleid attribuut 'Leeftijd'

Je kan jezelf de vraag stellen, waarom voegen we deze toe? Het antwoord is dat we compleet willen zijn als we bepaalde vragen krijgen van de gebruikers. Als we een expliciete vraag hebben ontvangen om leeftijd toe te voegen, kunnen we ook tonen dat dit uit het model af te leiden is en hierbij de gebruikers gerust stellen. Tegelijk voorkomen we op deze manier dat er redundantie gecreëerd wordt, en alle problemen die hierbij komen kijken. In het bovenstaande voorbeeld voorkomen we ook dat we continue de leeftijden moeten aanpassen telkens iemand verjaart.

Een afgeleid attribuut kan trouwens ook afgeleid worden op basis van attributen van andere entiteittypes. Het attribuut 'Aantal opgenomen studiepunten' van een entiteit van het entiteittype 'Inschrijving' kan je afleiden uit het attribuut 'Aantal studiepunten' van de entiteiten van het entiteittype 'Opleidingsonderdeel' dat opgenomen zijn binnen een inschrijving. De ERD notatie laat echter niet toe aan te geven wat de berekening is achter een afgeleid attribuut, dat moet je oplossen met wat extra documentatie of een nota op je model.

3.6.3 Samengestelde en enkelvoudige attributen

Een attribuut kan samengesteld of enkelvoudig zijn. Samengestelde attributen kunnen nog opgedeeld worden in verschillende attributen. Enkelvoudige attributen kunnen niet meer opgedeeld worden.

Een voorbeeld van een samengesteld attribuut van het entiteittype 'Persoon' is 'Adres'. Een adres bestaat namelijk uit een straat, huisnummer, ... We zouden met andere woorden het attribuut 'Adres' ook kunnen vervangen door de attributen 'Straat', 'Huisnummer', ...

In een ERD gaan we het onderscheid tussen samengesteld en enkelvoudig niet aanduiden. Er is dus geen verschil in notatie.

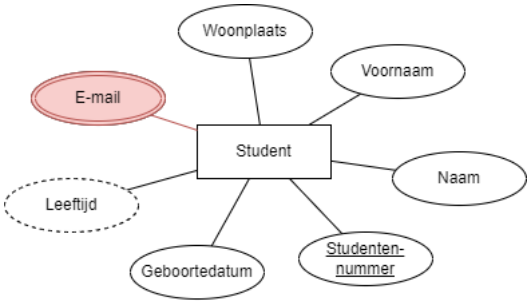
De keuze tussen samengestelde of enkelvoudige attributen is een afweging tussen eenvoud en leesbaar enerzijds en correct en volledig anderzijds. Vaak geven we de voorkeur aan correct en volledig.

3.6.4 Enkelwaardige en meerwaardige attributen

Een attribuut zal typisch voor een entiteit maximaal 1 waarde bevatten. Bijvoorbeeld voor het entiteittype 'Student' zal voor de entiteit 'Sam Peters' het attribuut 'Voornaam' de waarde 'Sam' bevatten. Voor elke entiteit hebben we voor een bepaald attribuut dus maximaal één waarde. We spreken in dit geval over een enkelvoudig attribuut.

In tegenstelling tot een enkelvoudig attribuut, kunnen we ook meervoudige attributen definiëren waarbij voor een attribuut van een entiteit meerdere waarden kunnen bestaan. Bijvoorbeeld voor het entiteittype 'Student' kan voor de entiteit 'Sam Peters' het attribuut 'E-mailadres' de waarden 'sam.peters@gmail.com' en 'huppeldepup@gmail.com' bevatten.

Een meervoudig attribuut wordt in een ERD aangeduid door middel van een dubbel omlijnde ellips waarin de naam van het attribuut wordt geplaatst.



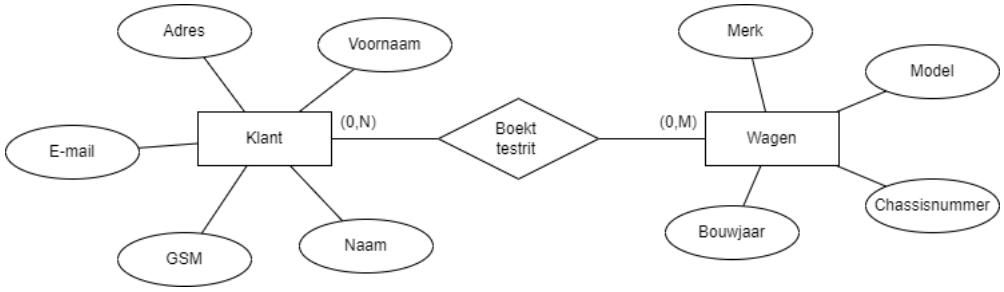
Figuur 3.19 Een voorbeeld van een entiteittype 'Student' met een meerwaardig attribuut 'E-mail'.

3.6.5 Attribuut van een relatie

In bepaalde N-M relaties merk je dat je bepaalde attributen niet kan plaatsen bij de entiteittypes.

Laten we even het volgende voorbeeld bekijken. In een bedrijf dat tweedehandswagens verkoopt, is het mogelijk voor klanten om een testrit te reserveren met één van de beschikbare wagens. Het is zelfs mogelijk dat een klant die interesse heeft in een wagen, meerdere keren een testrit doet met dezelfde wagen. Daarnaast kunnen voor een wagen verschillende klanten testritten inboeken. Als we dit zouden omzetten naar een ERD komen we op het volgende diagram:

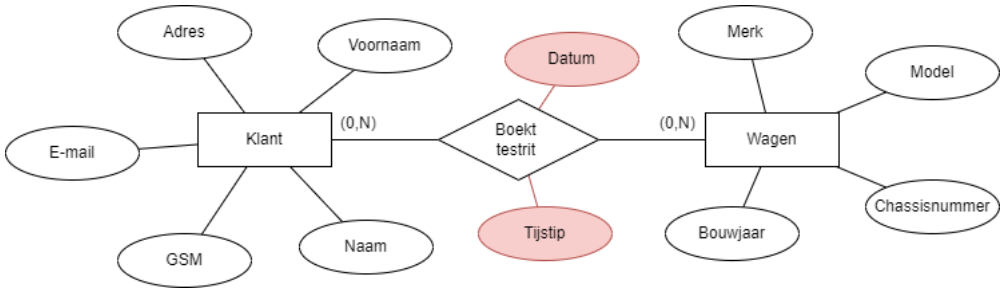
Onze opdrachtgever wil ook weten op welke dag en tijdstip de testrit ingepland is, dus we willen de attributen 'Dag' en 'Tijdstip' toevoegen. Mochten we deze attributen toevoegen bij het entiteittype 'Auto', betekent dat we voor elke entiteit van 'Auto' slechts één dag en één tijdstip kunnen vastleggen. Maar aangezien meerdere personen een testrit kunnen aanvragen, kunnen voor één wagen verschillende momenten vastgelegd zijn. We zouden van dit als een meerwaardig attribuut kunnen maken van het entiteittype 'Auto', maar dan weten we ook niet meer welke dag en tijdstip behoren tot welke klant.



Figuur 3.20 Een voorbeeld van een N-M relatie met een relatie 'Boekt testrit' tussen de entiteittypes 'Klant' en 'Wagen'.

Mochten we de attributen toevoegen aan het entiteittype 'Klant', dan zitten we met een gelijkaardig probleem. Een klant kan met meerdere wagens een dag en tijdstip vastleggen voor een testrit. Mochten we dit dus als een attribuut toevoegen aan het entiteittype 'Klant', dan zou voor alle vastgelegde testritten hetzelfde datum en tijdstip gelden. We kunnen de attributen 'Dag' en 'Tijdstip' dus niet toevoegen bij het entiteittype 'Klant'.

De oplossing is om deze attributen als eigenschappen van de relatie te beschouwen. Hierdoor gaan we voor elke combinatie van klant en auto aangeven op welke dag en tijdstip de testrit gepland is.



Figuur 3.21 Een voorbeeld van een N-M relatie met een relatie 'Boekt testrit' tussen de entiteittypes 'Klant' en 'Wagen'. De relatie is uitgebreid met twee attributen 'Datum' en 'Tijdstip'.

Opgelet! Een attribuut van een relatie kan enkel bestaan indien het gaat om een N-M relatie. Voor alle andere types van relaties is dit niet mogelijk. We kunnen de attributen dan altijd aan één van de entiteittypes verbinden.

3.6.6 Oefening 3

Ga naar oefening 3 van 'Conceptueel datamodel: oefeningen' en zet de beschrijvingen stap voor stap om in een ERD.

3.6.7 Oefening 1 - Verdergezet

Ga terug naar oefening 1 van 'Conceptueel datamodel: oefeningen' en verfijn jouw resultaat op basis van het bovenstaande. Komt de notatie van jouw attributen overeen met het type? Zijn er bepaalde relaties die een attribuut hebben?

Breid jouw ERD uit met jouw relaties.

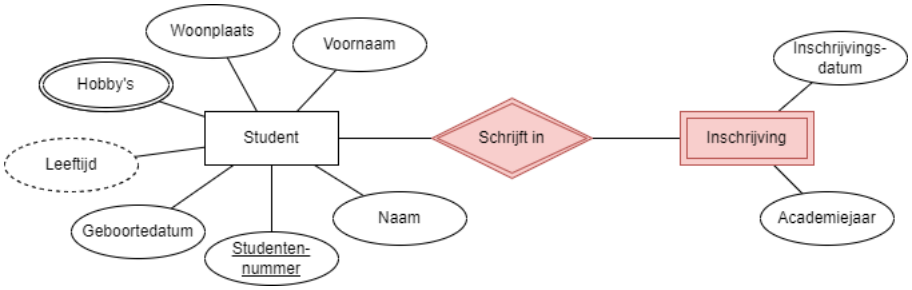
3.7 Zwakke en sterke entiteiten

Binnen de entiteittypes kunnen we nog een onderscheid maken tussen **zwakke entiteittypes** en sterke entiteittypes (ook wel gewone entiteittypes genoemd). Een zwak entiteittype beschrijft een verzameling van entiteiten die niet op zichzelf kunnen bestaan. De entiteiten zijn afhankelijk van het bestaan van andere entiteiten.

Een voorbeeld van een zwak entiteittype is een 'Inschrijving' aan de UCLL Hogeschool. Een inschrijving kan niet bestaan als er geen student bestaat. Er moet met andere worden al een entiteit 'Sam Peters' van entiteittype 'Student' aangemaakt zijn vooraleer we Sam kunnen inschrijven. De entiteit van het entiteittype 'Inschrijving' zal een verwijzing bevatten naar de entiteit van het entiteittype 'Student'. In het ERD moet er dus een relatie bestaan tussen beide entiteittypes.

Voor de identificatie van de zwakke entiteit gaan we gebruik maken van het sleutelattribuut van de sterke entiteit. Op basis van bovenstaand voorbeeld, gaan we dus gebruik maken van het studentnummer van 'Sam Peters' om de inschrijving (deels) te identificeren. Aangezien Sam meerdere inschrijvingen kan hebben over verschillende jaren heen, is het studentnummer onvoldoende om de inschrijving te identificeren. We gaan die combineren met een volgnummer, of academiejaar, ...

Een zwak entiteittype wordt in een ERD aangeduid als een vierkant met een dubbele omlijning met hierin de naam van het entiteittype. De relatie tussen het zwak en sterk entiteittype wordt weergegeven als een ruit met dubbele omlijning.



Figuur 3.22 Een voorbeeld van een zwak entiteittype'Inschrijving' met relatietype 'Schrijft in'.

Let op het feit dat we in het ERD het identificerend attribuut van het entiteittype 'Student' niet overnemen als identificerend attribuut van 'Inschrijving'. We kunnen dit wel afleiden uit de notatie met de dubbel omlijnde rechthoeken en ruiten.

In de praktijk kan een zwakke entiteit wel een eigen sleutelattribuut krijgen bij het uittekenen van een logisch of fysiek datamodel omdat dit efficiëntiewinst oplevert.

3.7.1 Oefening 4

Ga naar oefening 4 van 'Conceptueel datamodel: oefeningen' en zet de beschrijvingen stap voor stap om in een ERD.

3.7.2 Oefening 5

Ga naar oefening 5 van 'Conceptueel datamodel: oefeningen' en zet de beschrijvingen stap voor stap om in een ERD.

3.8 Extended Entity Relationship Diagram (EERD)

Het ERD werd uitgebreid met een aantal concepten om meer complexe versies van de realiteit te kunnen modelleren.

3.8.1 Subtypes en supertypes

Wat zijn subtypes en supertypes?

Een entiteittype is een subtype van een ander entiteittype indien het eerste entiteittype alle attributen en relaties overerft van het tweede entiteittype en dit uitbreidt met eigen attributen en relaties. Het tweede entiteittype is een meer generiek entiteittype met een aantal algemene attributen en relaties, en noemen we het supertype.

Een voorbeeld van subtypes en supertypes is bijvoorbeeld een supertype 'Werknemer', en twee subtypes 'Tijdelijke werknemer' en 'Permanent werknemer'. Van het entiteittype 'Werknemer' houden we de typische eigenschappen bij: naam, voornaam, geboortedatum, werknemersnummer, ... Daarnaast hebben we het entiteittype 'Tijdelijke werknemer' dat alle attributen en relaties overerft van het entiteittype 'Werknemer', maar dit uitbreidt met de attributen 'Startdatum contract' en 'Einddatum contract'. Het entiteittype 'Permanente werknemer' breidt het entiteittype 'Werknemer' uit met de attributen 'Startdatum werknemer' en 'Aantal jaren dienst'.

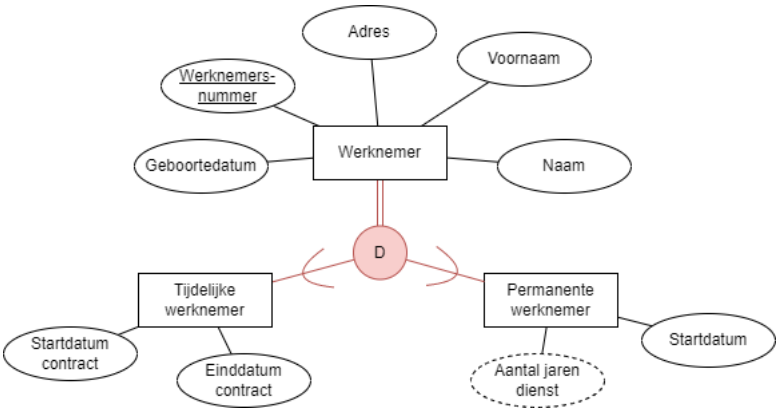
Subtypes kunnen overlappend of disjunct zijn. Bij overlap kan een entiteit van het supertype tegelijk ook een entiteit zijn van **meerdere subtypes**. Bij disjunct, kan een entiteit van het supertype tegelijk een entiteit zijn van **hoogstens één subtype**. Bovenstaande voorbeeld is disjunct, aangezien een werknemer of een tijdelijke werknemer is, of een permanente werknemer maar nooit beide.

De overerving kan totaal of optioneel zijn. Bij totale overerving moet een entiteit van het supertype ook een entiteit zijn van minstens één van de subtypes. Bij optionele overerving mag een entiteit van het supertype een entiteit zijn van minstens één van de subtypes, maar is dit niet verplicht. Bovenstaande voorbeeld is totaal, elke werknemer moet of een tijdelijke medewerker zijn, of een permanente medewerker zijn.

Elke entiteit van een subtype, is ook een entiteit van het supertype. De relatie tussen een subtype en supertype is een 1-1 relatie. Het sleutelattribuut van het supertype, wordt ook het sleutelattribuut van de subtypes. Dit wordt geïmpliceerd door de overerving, en moet dus niet toegevoegd worden.

Notatie ERD

De notatie van sub- en supertypes gebeurt aan de hand van een cirkel met hierin een 'O' om overlappende subtypes aan te geven, of 'D' om disjuncte subtypes aan te geven. Vanuit de subtypes gaat een lijn naar de cirkel waarop we een boog tekenen. Vanuit de cirkel gaat een lijn naar het supertype. De lijn wordt dubbel of enkel getekend. Een dubbele lijn duidt op een totale overerving, een enkele lijn op een optionele overerving.



Figuur 3.23 Een voorbeeld van een supertype 'Werknemer' met twee subtypes 'Tijdelijke werknemer' en 'Permanente werknemer'.

Generalisatie en specialisatie

Het proces om subtypes en supertypes te definiëren, wordt ook wel generalisatie en specialisatie genoemd. Generalisatie verwijst naar het proces waarbij we voor een aantal gelijkaardige

entiteittypes een supertype creëren waarin we de gemeenschappelijke attributen en relaties verzamelen, en waarvan de initiële entiteittypes van overerven. Specialisatie start van een entiteittype waarbij verschillende subtypes worden aangemaakt die elk een aantal eigen attributen of relaties hebben die verschillend zijn voor de andere subtypes. Met andere woorden, het is de richting die bepaalt of je spreekt over generalisatie of specialisatie, maar het resultaat is hetzelfde.

Wanneer toepassen?

Subtypes en supertypes zijn handig wanneer er een grote overlap is tussen een aantal entiteittypes. Het zorgt voor een overzichtelijk schema en reikt ook al oplossingen naar de ontwikkeling van de tabellen.

3.8.2 Oefening 6

Ga naar oefening 6 van ‘Conceptueel datamodel: oefeningen’ en zet de beschrijvingen stap voor stap om in een ERD.

4

Logisch datamodel

4.1 Doel

In het volgende hoofdstuk willen we het concept databankmodel introduceren als inleiding om het logische datamodel kunnen op te bouwen. Van daaruit kunnen we dan de omzetting van het conceptueel naar het logisch datamodel overlopen. We bekijken:

- een databankmodel
- de rol van een databankmodel
- het relationeel databankmodel
- omzetten van een conceptueel datamodel naar een logisch datamodel volgens het relationeel databankmodel
- Jullie moeten de volgende termen kunnen uitleggen, hoe we dit weergeven (waar relevant) en toelichten aan de hand van een voorbeeld:
  - Logisch datamodel
  - Databankmodel
  - Relationeel databankmodel
  - Tabel
  - Sleutel (waaronder kandidaatsleutel, primaire sleutel, vreemde sleutel, samengestelde sleutel, natuurlijke sleutel, technische of surrogaatsleutel)
  - Normaliseren

4.2 Inleiding

Na het conceptueel datamodel gaan we verder met het ontwerp van onze databank. We hadden een databank omschreven als een verzameling van 'persistente data'. Om te bepalen wat er specifiek in die verzameling opgenomen moet worden, hebben we het conceptueel datamodel opgesteld die de vraag 'Welke informatie moeten we in onze databank opnemen' beantwoordt. Daarbij wordt de verzameling afgebakend tot een bepaald thema of toepassing. Een voorbeeld hiervan is de databank van de UCLL Hogeschool studentenadministratie, met data betreffende studenten, opleidingen, inschrijvingen, ...

In een volgende stap gaan we nadenken over de **structuur** van deze verzameling van data. We gaan nadenken hoe we de data kunnen organiseren. En data kan op verschillende manieren georganiseerd worden.

Vooraleer we verder focussen op de verschillende manieren waarom we structuur kunnen toepassen, moeten we even stil staan bij met de vraag **'Waarom is structuur belangrijk?'**. Wanneer we met machines werken, zorgt structuur ervoor dat we deze computers kunnen aanleren om binnen het kader van deze structuur te werken. De structuur op zich zorgt voor voorspelbaarheid zodat het makkelijk wordt om hier procedures op te definiëren. Dit geldt trouwens niet alleen voor computers, het feit dat er markeringen zijn op de wegen structureert de wegen op zo'n manier dat er regels kunnen worden bepaald die de verkeersveiligheid bevorderen. En daar zijn we allemaal blij om.

Voor data geeft dit dan nog eens het bijkomende voordeel dat structuur op zich bijkomende informatie bevat (we noemen dit metadata - data over de data - we verzinnen dit niet) die dan gebruikt kan worden voor het bepalen van procedures. Zodoende kunnen we de toegang tot de data beheren, opslag optimaliseren, standaardprocedures ontwikkelen, ...

Net zoals dit geldt voor heel wat andere domeinen, kan je eenzelfde concept op heel wat verschillende manieren gaan structureren. Voor data is dit niet anders, ook data kan op verschillende manieren worden georganiseerd. Je moet dan de volgende vragen stellen:

- over welk type van data spreken we?
- wat wil je met deze data gaan doen?

Omdat IT-ers nogal houden van standaarden (en jullie als toekomstige IT-ers ondertussen enthousiast moeten worden omdat er duidelijk een standaard aan zit te komen), zou het geen verrassing mogen zijn dat ze ook al heel wat standaarden hebben bepaald voor datamodellen en om data te structureren. We noemen deze standaarden voor databankmodellen.

4.3 Databankmodellen

Een databankmodel bepaalt de structuur van een databank en hoe de data binnen een databank wordt georganiseerd en gemanipuleerd. Daar zitten wel enkele elementen in waarop we wat dieper moeten ingaan.

Eerst en vooral zal een databankmodel data op een specifieke manier gaan **structureren**. Daarbij wordt typisch gebruik gemaakt van een aantal componenten of bouwblokken die elk een rol spelen in de structuur van het datamodel. Die rol wordt dan typisch beschreven in een aantal principes of regels die de samenwerking van de verschillende componenten vastlegt.

De standaarden rond databankmodellen (en andere domeinen) geven nog een aantal belangrijke voordelen die we zeker niet uit het oog mogen verliezen:

- Eerst en vooral gaat het om standaarden die **over heel de industrie** gebruikt worden. Wanneer jullie het in jullie professionele loopbaan over een bepaalde standaard hebben, begrijpt iedereen rond de tafel met een IT opleiding waarover je het hebt, dit geldt ook voor databankmodellen.

- Een tweede voordeel is dat heel wat onderzoekers blijven verder werken op deze standaarden en bijkomende uitbreidingen (herinner EERD) en procedures ontwikkelen waar iedereen baat bij heeft, dus standaarden blijven typisch wel **evolueren**. Jullie taak zit er in om bij te blijven.

- Een derde voordeel is dat de standaarden een basis vormen voor bedrijven om **software** producten te ontwikkelen. Aangezien ze gebaseerd zijn op standaarden kunnen ze op hun beurt goed samenwerken met andere producten die dan ook weer standaarden gebruiken.

Over de jaren heen zijn er heel wat verschillende databankmodellen ontwikkeld. Sommige zijn ontwikkeld vanuit bepaalde technologieën. Anderen om met nieuwe types van data te kunnen werken. Anderen dan weer omdat de ontwikkeling van al maar krachtigere machines de mogelijkheden om met data te werken systematisch uitbreiden. Kortom de horizon van het mogelijke breidt constant uit, en zo ook het aanbod aan beschikbare databankmodellen.

Van al deze databankmodellen gaat er één specifiek databankmodel al een hele tijd mee. Het relationeel databankmodel werd ontwikkeld in de jaren 70 door Codd en geldt vandaag nog steeds al een standaard binnen de IT industrie. Binnen dit vak is dit dan ook het databankmodel waarop we verder de focus leggen, de andere komen aan bod in andere vakken.

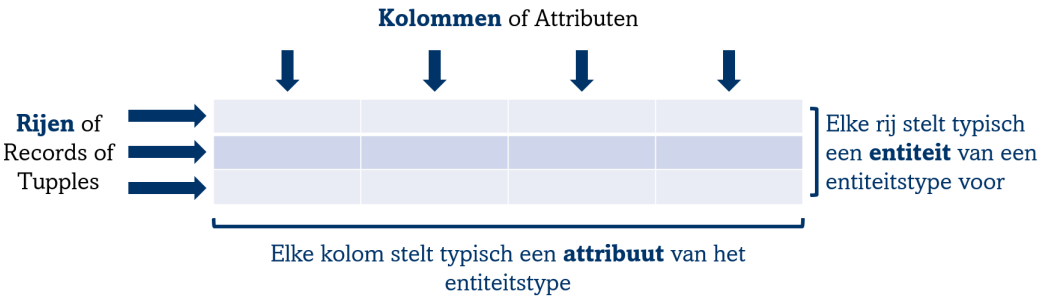
4.3.1 Relationeel databankmodel

Tabel

Het relationeel databankmodel maakt gebruik van één component of bouwblok, de **tabel**. Alle data die we willen bijhouden, zullen we in één of meerdere tabellen gieten. Elke tabel stelt algemeen genomen een verzameling gelijkaardige data voor, bijvoorbeeld een tabel met alle gegevens van studenten. Een tabel is gestructureerd als een 2-dimensionale matrixstructuur, waarbij we de horizontale dimensie de rijen noemen en de verticale dimensie de kolommen.

Een **rij** stelt een record (of entiteit) voor van de tabel die we dan vervolgens gaan beschrijven. Een tabel kan theoretisch een oneindig aantal rijen bevatten. Mochten we een tabel creëren met studentengegevens, zou elke rij een student kunnen voorstellen.

Een **kolom** stelt een attribuut (of eigenschap) voor van de tabel die telkens een stukje beschrijving van de rij zal bevatten. Een tabel bevat een vast aantal kolommen. Mochten we een kolom willen toevoegen, wat op zich wel mogelijk is, moeten we deze kolom invullen voor elk van de rijen die in de tabel opgenomen zijn. In ons voorbeeld van een tabel met studentengegevens, zou elke kolom een bepaalde eigenschap van de student bevatten, bijvoorbeeld de naam, de geboortedatum, ...



Figuur 4.1 Een tabel als een 2-dimensionale matrix structuur bestaande uit rijen en kolommen.

Omdat een tabel een specifieke structuur heeft van kolommen, zal dus ook alle data die in één tabel wordt bijgehouden, altijd dezelfde structuur hebben. Telkens we bijkomende data met een andere structuur bijhouden, creëren we een bijkomende tabel. In een relationeel databankmodel gaan we het geheel van een aantal tabellen een databank noemen. Onze relationele databank kan dus heel wat verschillende tabellen gaan bevatten, om deze tabellen dan vervolgens met elkaar te combineren en verbanden te creëren, maken we gebruik van sleutels.

Sleutels

Een sleutel bestaat uit één of meerdere kolommen (attributen) van een tabel die voldoet aan de volgende drie voorwaarden:

- De waarde van de combinatie van één of meerdere kolommen in een rij vormt een **unieke identificatie** van de volledige rij. Voor elke rij moet de waarde van de sleutel dus uniek zijn, anders gezegd mogen twee rijen in de tabel niet eenzelfde sleutel hebben.
- De combinatie van de kolommen die de sleutel vormen moet **minimaal** zijn. Daarmee willen we aangegeven dat indien we in de combinatie één van de kolommen zouden verwijderen, we niet meer zouden voldoen aan de eerste voorwaarde. Indien we een sleutel hebben die bestaat uit één kolom, is onze sleutel per definitie minimaal.
- De waarde van de sleutel mag ook niet leeg zijn, de sleutel moet dus altijd ingevuld zijn.

Binnen het relationeel model bepalen we **voor elke tabel één primaire sleutel** (primary key). De waarde van de primaire sleutel vormt voor elke een unieke identificatie. Deze primaire sleutel zullen we gebruiken om de verschillende tabellen met elkaar te gaan verbinden.

De primaire sleutel wordt gekozen uit een aantal kandidaatsleutels. Een **kandidaatsleutel** (candidate key) is een potentiële sleutel voor een tabel, en die dus ook voldoet aan de drie voorwaarden waaraan een sleutel moet voldoen. Een tabel kan dus meerdere sleutels hebben. Vanuit de lijst van kandidaatsleutels kiezen we één primaire sleutel. Om een goede sleutel te kiezen vanuit de lijst van mogelijke sleutels kan je de volgende criteria in rekening nemen:

- Een sleutel moet zo **stabiel** mogelijk zijn, we gaan er zelf vanuit dat een sleutel eigenlijk nooit aangepast wordt. Hier zijn enkele uitzonderingen op, maar we proberen een veranderende primaire sleutel ten alle koste te vermijden. Bijvoorbeeld, indien je binnen de context van studentenadministratie de keuze hebt tussen een rijksregisternummer of een studentennummer om een student uniek te identificeren, kies je een studentennummer. Niet enkel hebben we binnen de studentenadministratie meer controle over het studentennummer, we kunnen er over waken dat dit nooit aangepast wordt. Een rijksregisternummer kan wel veranderen. Daarbij is het gebruik van een rijksregisternummer in een databank (en erbuiten) onderhevig aan heel strikte GDPR wetgeving, maar dat leggen we in een ander vak uit.
- Een sleutel omvat ideaal **zo weinig mogelijk kolommen**. Sleutels kunnen uit één of meerdere kolommen bestaan. Een sleutel die bestaat uit meerdere kolommen, noemen we een *samengestelde sleutel* simpelweg omdat dit samengesteld wordt uit meerdere waarden. We vermijden samengestelde sleutels omdat dat het geheel wat complexer maakt, en complexiteit brengt een kost met zich mee die we trachten te vermijden. Complexiteit betekent dat het geheel moeilijk te begrijpen is voor de betrokken partijen, mogelijk een verlies aan performantie bij het verwerken van de data, enz. Van zodra jullie in andere vakken bijleren rond performantie, automatisatie, security, ... in het kader van databanken zullen we meer in staat zijn om de kost van complexiteit toe te lichten.
- Een sleutel is idealiter een **eenvoudige waarde**, bijvoorbeeld een nummer of een korte opeenvolging van karakters. Van zodra het complexer wordt, of zelfs speciale karakters gaat gebruiken, wordt het risico op fouten groter.

- Een sleutel wordt idealiter genomen uit de **context** van datgene wat we beschrijven. Een sleutel die zowel waardevol is binnen de databank (als primaire sleutel) maar ook buiten het systeem gebruikt wordt, heeft als voordeel dat heel wat betrokken personen snel begrijpen waarover het gaat. Bijvoorbeeld, een studentennummer is ook buiten de databank een manier om studenten te identificeren. Dus mogelijk een goede primaire sleutel. Een sleutel die zowel gebruikt wordt binnen de databank als erbuiten, noemen we een **natuurlijke sleutel** (natural key).

Indien we geen kandidaatsleutels hebben, of de beschikbare kandidaatsleutels zijn geen geschikte keuze, hebben we altijd de mogelijkheid om zelf een sleutel te definiëren. We noemen die een *technische sleutel* of *surrogaatsleutel*. De technische sleutel wordt enkel binnen de databank gebruikt, en heeft buiten de databank geen waarde. Het heeft dan als effect dat we een bijkomende kolom bijmaken die de sleutel zal bevatten.

Uiteindelijk moet elke tabel een primaire sleutel hebben. We hadden al aangegeven dat de primaire sleutels het mogelijk gaan maken om verbanden te creëren tussen de tabellen. Dit gebeurt door een uitwisseling van primaire sleutels. Het mechanisme van de uitwisseling wordt toegelicht in 'Van conceptueel naar logisch datamodel', maar het zal erin resulteren dat de primaire sleutel van de ene tabel toegevoegd wordt aan een andere tabel. Van zodra we een primaire sleutel toevoegen aan een andere tabel, en dus een kolom toevoegen aan die tabel, noemen we deze sleutel een **vreemde sleutel** (foreign key).

Het gebruik van de primaire sleutels en vreemde sleutels staat centraal in het relationeel model. Het is de basis om de relaties vast te leggen en te definiëren.

4.3.2 Andere databankmodellen

Naast het relationele databankmodel, zijn er ook heel wat andere databankmodellen, elk met hun specifieke toepassing. Bepaalde van deze databankmodellen hebben hun weg gevonden in bepaalde technologieën zodat een goed begrip van de modellen, jullie in staat moet stellen om de werking van de technologie te begrijpen. We gaan in hier in andere vakken dieper op in, maar om jullie alvast een voorsmaakje te geven, hebben we alvast enkele databankmodellen opgelijst:

- Hiërarchisch datadankmodel: maakt gebruik van boom-structureren om data voor te stellen
- Netwerk databankmodel: maakt gebruik van netwerk-structureren om data voor te stellen
- Dimensionaal databankmodel: gebruik in de context van data warehouses
- Object-georiënteerd databankmodel: combineert de mogelijkheden van een relationeel met het object-georiënteerd programmeren
- Graaf databankmodel: maakt gebruik van graaf-structureren om data voor te stellen

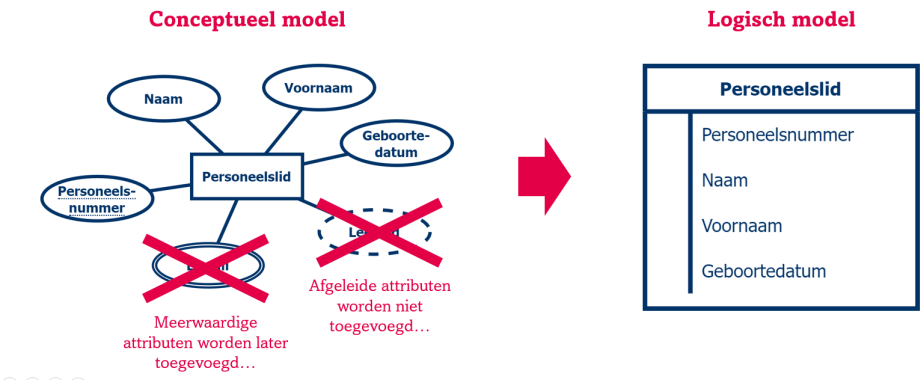
4.4 Van conceptueel naar logisch datamodel

Het logisch datamodel beantwoordt de vraag 'Hoe moeten we de data structureren volgens ons gekozen databankmodel?'. We hebben een conceptueel datamodel ontwikkeld, en de keuze gemaakt voor een relationeel databankmodel. De omzetting van dit conceptueel datamodel verloopt in een aantal stappen.

4.4.1 Stap 1: Entiteiten worden tabellen

In de eerste stap creëren we een tabel voor elk van de entiteittypes. Deze tabel geven we dezelfde naam als het entiteittype. Vervolgens definiëren we de kolommen van de tabel, waarbij we voor elk attribuut een kolom aanmaken, met uitzondering van de meerwaardige attributen en de afgeleide attributen. We komen later nog terug op de meerwaardige attributen. De naam van de gecreëerde kolom komt overeen met de naam van het attribuut.

De notatie van het logisch datamodel is verschillend van dat van het conceptueel datamodel. We geven het geheel van een tabel weer als een kader, met hierin een lijst van de attributen.

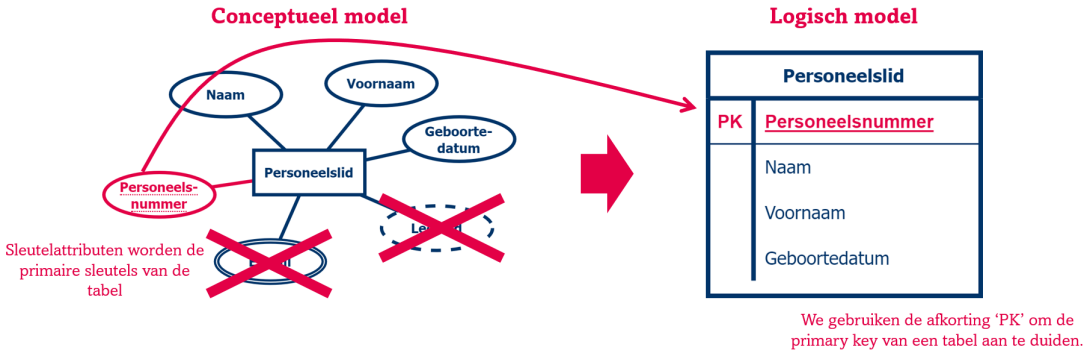


Figuur 4.2 Stap 1: we zetten de entiteittypes om naar tabellen.

4.4.2 Stap 2: Elke tabel krijgt een primaire sleutel toegewezen

In een tweede stap gaan we voor elke tabel een primaire sleutel aanduiden. Dat betekent dat we een aantal kandidaatsleutels moeten bepalen en vanuit deze lijst vervolgens de beste kandidaat selecteren.

We duiden de primaire sleutel aan door de kolommen van de sleutel bovenaan de lijst te plaatsen met hieronder een lijn om duidelijk een onderscheid te maken tussen de primaire sleutel en de resterende kolommen. Daarbij zetten we ook voor de verschillende kolommen van de primaire sleutel de code 'PK' (van primary key) en om volledig zeker te zijn dat niemand zich kan vergissen, onderlijnen we de kolomnamen ook nog eens.



Figuur 4.3 Stap 2: we bepalen voor elke tabel een primaire sleutel.

We doen dit voor elke tabel. Houd ook in het achterhoofd dat je een technische sleutel mag aanmaken, mocht het moeilijk zijn om met de bestaande kolommen een primaire sleutel te maken.

4.4.3 Stap 3: Voor elke 1-1 en 1-N relaties zetten we de primaire sleutel over als vreemde sleutel

In de derde stap gaan we al enkele van de relaties vanuit het conceptueel datamodel omzetten naar een equivalent in het logisch datamodel. Aangezien we binnen het logisch datamodel verbanden creëren door middel van sleutels, gaan we dus hiermee aan de slag.

Voor elke **1-N relatie** gaan we de primaire sleutel van de 1-zijde overdragen naar de N-zijde. Aan de N-zijde wordt deze primaire sleutel dan een vreemde sleutel. Elke rij van de tabel aan de N-zijde zal dus een bijkomende kolom krijgen met hierin een verwijzing (door middel van de primaire sleutel) naar een rij in de tabel van de 1-zijde.

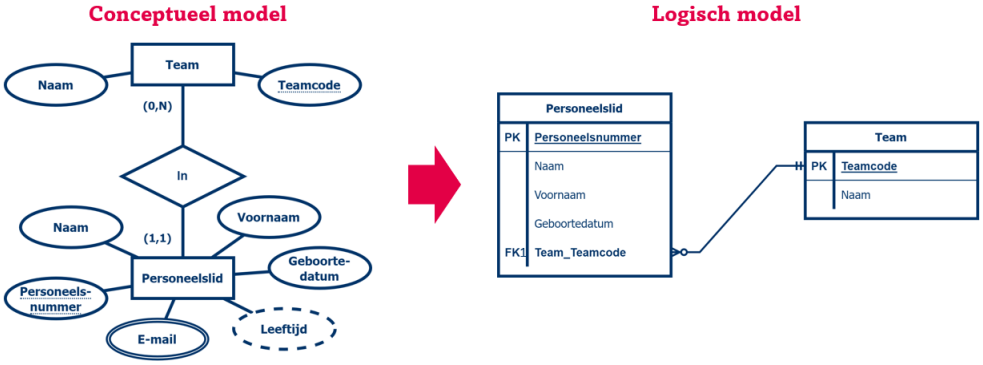
Bijvoorbeeld, we hebben twee entiteittypes: 'Team' en 'Personeelslid'. Elk personeelslid zit in één en exact één team. Elk team kan meerdere personeelsleden omvatten. We creëren twee tabellen, enerzijds een tabel 'Team' en anderzijds een tabel 'Personeelslid'. De tabel 'Team' krijgt als primaire sleutel 'Teamcode' waarbij elk team in de tabel een unieke teamcode krijgt. De tabel 'Personeelslid' krijgt als primaire sleutel 'Personeelsnummer' waarbij elk personeelslid in de tabel een uniek personeelsnummer krijgt. Om het verband tussen de twee entiteittypes aan te duiden in het logisch datamodel, gaan we de teamcode vanuit de 'Team' tabel (1-zijde) toevoegen aan elk personeelslid in de 'Personeelslid' tabel (N-zijde).

Omgekeerd is dit geen optie, we kunnen aan een team slecht één waarde toevoegen. Mochten we dus een verwijzing maken vanuit het team naar één personeelslid, zijn we het verband met de andere teamleden van het team kwijt.

In het datamodel voegen we de vreemde sleutel toe aan de lijst van kolommen. We duiden de vreemde sleutel aan met de afkorting 'FK' (van foreign key) gevolgd door een volgnummer, bijvoorbeeld FK1, FK2, ... Het volgnummer moet het mogelijk maken om de verschillende

vreemde sleutels van elkaar te onderscheiden. Indien we een vreemde sleutel hebben die uit meerdere kolommen bestaat, zetten we elk van de kolommen over. We geven ze dan allemaal hetzelfde volgnummer om aan te duiden dat ze samen horen.

Het verband duiden we nog steeds aan door een lijn waarbij we de kraaienpootnotatie gebruiken. De kraaienpootnotatie is verschillend van de (min,max) notatie die jullie tot nu toe gebruikt hebben. De lijn is puur informatief en dient om de leesbaarheid van het datamodel te verhogen.



Figuur 4.4 Stap 3: voor elke 1-N relatie zetten we de primaire sleutel over als vreemde sleutel

Voor elk van de **1-1 relaties** doen we iets gelijkaardigs, alleen kunnen we hier kiezen vanuit welke zijde we de primaire sleutel nemen om aan de andere zijde als vreemde sleutel toe te voegen.

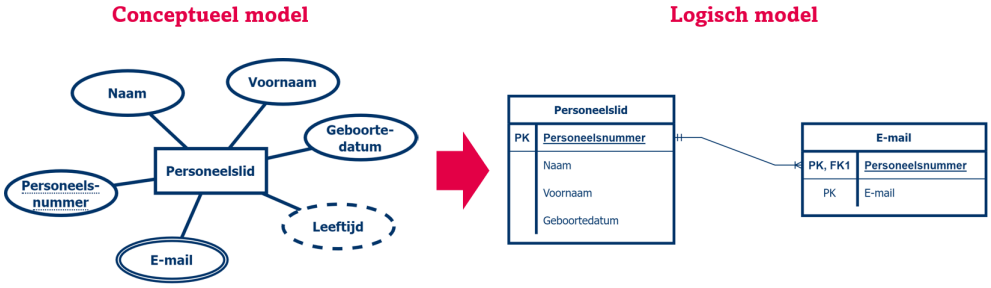
De richtlijn die we daar kunnen meegeven om jullie te begeleiden in het maken van een keuze, is om naar de minimumkardinaliteit te kijken. Indien een bepaald entiteittype niet kan bestaan zonder de relatie (minimumkardinaliteit is dus ‘1’), neemt deze tabel best de primaire sleutel van de andere tabel op als vreemde sleutel. Aangezien een entiteit enkel bestaan indien deze een relatie heeft, zal de rij die de entiteit beschrijft dus altijd deze waarde ingevuld moeten hebben. Want zonder deze waarde, bestaat de relatie niet.

Indien beide entiteitstypes niet zonder elkaar kunnen bestaan, beide hebben dus een minimumkardinaliteit van ‘1’, of beide kunnen allebei zonder elkaar bestaan, beide hebben dus een minimumkardinaliteit van ‘0’, mag je kiezen. Typisch kies je dan voor de tabel met het kleinste aantal rijen.

Meerwaardige attributen

Meerwaardige attributen zijn op zich 1-N relaties waarbij een entiteit gerelateerd wordt aan meerdere attribuutwaarden. Daarom moet je hiervoor een gelijkaardige aanpak toepassen als bij de omzetten van de 1-N relaties. Voor elk meerwaardig attribuut maak je een nieuwe tabel met als naam de naam van het meerwaardig attribuut. Aan deze tabel voeg je de waarde van het attribuut toe en dan vervolgens de primaire sleutel van de tabel van het entiteittype

als vreemde sleutel. Vergeet dat niet om een primaire sleutel aan te duiden en een lijn in kraaienpootnotatie toe te voegen.



Figuur 4.5 Stap 3: voor elk meerwaardig attribuut creëren we een bijkomende tabel

Bijvoorbeeld, het meerwaardig attribuut 'E-mail' van het entiteittype 'Personeelslid' resulteert in een tabel 'E-mail' met hierin een kolom 'E-mail'. Om elk emailadres te kunnen verbinden aan een personeelslid, voegen we de primaire sleutel van de tabel 'Personeelslid' toe als vreemde sleutel aan de tabel 'E-mail'.

4.4.4 Stap 4: Voor elke N-M relaties creëren we een tussentabel

Nu wordt het even complexer. We hebben alvast een oplossing gevonden voor de verschillende entiteittypes en hun attributen. Voor de 1-1 en 1-N relaties hebben we ook een oplossing gevonden. In een volgende stap leggen de focus op de N-M relaties. Om N-M relaties mogelijk te maken, moeten we verder gaan dan het eenvoudig uitwisselen van primaire sleutels omdat dit voor N-M relaties geen goede oplossing oplevert.

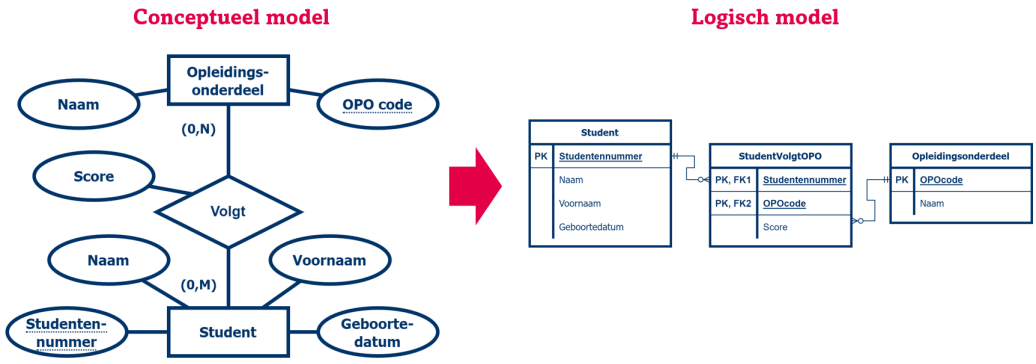
Laten we even starten vanuit een voorbeeld. We hebben het entiteittype 'Student' en het entiteittype 'Opleidingsonderdeel'. Tussen beide entiteittypes hebben we een relatie die uitdrukt dat een student in meerdere opleidingsonderdelen kan inschrijven, en omgekeerd dat in een opleidingsonderdeel meerdere studenten kunnen inschrijven. In eerste instantie maken we van beide entiteittypes twee tabellen, de tabel 'Student' en de tabel 'Opleidingsonderdeel'. De eerste tabel krijgt als primaire sleutel 'Studentennummer', de twee tabel krijgt als primaire sleutel 'OPOcode'. Mochten we de 'OPOcode' toevoegen aan de tabel 'Student', zou een student maar naar één opleidingsonderdeel kunnen verwijzen en dus maar één opleidingsonderdeel kunnen volgen, wat niet correct is. Omgekeerd, mochten we het 'Studentennummer' toevoegen aan de tabel 'Opleidingsonderdeel', zou een opleiding maar door één student kunnen worden opgenomen, wat ook niet correct is.

We lossen dit op door een tussentabel aan te maken, waaraan we voor elke combinatie waarbij een entiteit van 'Student' een entiteit van 'Opleidingsonderdeel' volgt een rij toevoegen. Elke rij bevat dan de primaire sleutel van de tabel 'Student' en de primaire sleutel van de tabel 'Opleidingsonderdeel'. De tabel zal dus telkens 2 vreemde sleutels bevatten. De combinatie van de twee vreemde sleutels, wordt dan de nieuwe primaire sleutel van de nieuwe tussentabel.

Dezelfde regels die we eerder gaven voor een sleutel, gelden hier ook. Dus indien tussen twee entiteiten meerdere relaties zijn, zou de combinatie van de vreemde sleutels mogelijk onvoldoende zijn om aan alle regels te voldoen. Dat betekent dat er mogelijk iets ontbreekt in je datamodel.

Bovenstaande voorbeeld resulteert met andere woorden in een tabel 'StudentVolgtOpleidingsonderdeel' waar we de kolom 'Studentennummer' en de kolom 'OPOcode'. Beide zijn vreemde sleutels, dus we duiden deze aan met de afkorting 'FK' (Foreign Key) gevolg door een volgnummer. De combinatie van beide sleutels is ook de primaire sleutel van de nieuwe tabel, dus we duiden elke kolom ook aan met afkorting 'PK' (Primary key).

Een N-M relatie is het enige type relatie dat zelf ook attributen kan hebben. In een logisch datamodel gaan we voor deze attributen van een relatie bijkomende kolommen creëren in de tussentabel. Mochten we dus in het bovenstaand voorbeeld voor elke combinatie tussen student en opleidingsonderdeel ook een score willen bijhouden, creëren we een bijkomende kolom 'Score' in de tabel 'StudentVolgtOpleidingsonderdeel'.



Figuur 4.6 Stap 4: we creëren een tussentabel voor elke N-M relatie

4.4.5 Stap 5: de speciale gevallen

Er resten nog enkele constructies in het conceptueel datamodel waar we nog geen oplossing voor hebben gegeven. We willen ze toch kort even toelichten zodat jullie ook deze kunnen omzetten vanuit een conceptueel naar een logisch datamodel.

Relaties van de graad 1

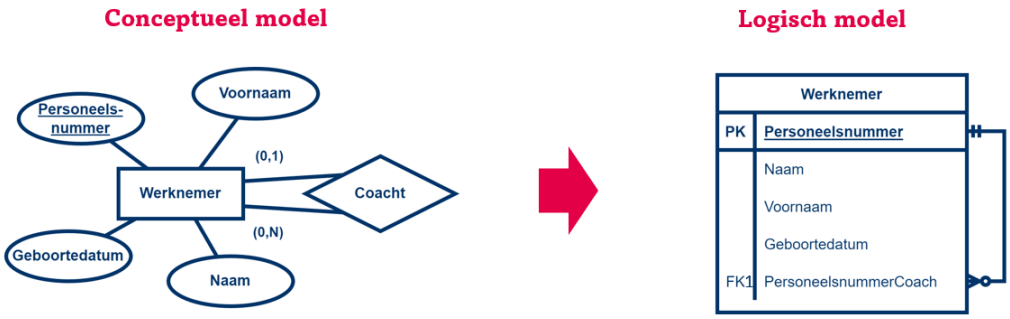
Een relatie van graad 1 in een conceptueel datamodel, ook wel een unaire relatie genoemd, is een situatie waarbij een entiteittype een relatie heeft met zichzelf.

Stel bijvoorbeeld een situatie waarbij een werknemer een andere werknemer coacht en we willen deze informatie opnemen in onze databank (en dus ook in ons datamodel). We hebben

dan een entiteittype ‘Werknemer’ die via een relatie ‘Coacht’ naar zichzelf verwijst. De relatie zegt dan vervolgens dat elke werknemer mogelijk meerdere andere werknemers coacht, maar mogelijk ook geen enkele. Daarnaast heeft elke werknemer maximaal 1 coach, maar ook dit is niet verplicht.

Om dit vervolgens om te zetten naar een logisch datamodel, behandelen we onze unaire relatie alsof het een binaire relatie is, wat overeenkomt met een 1-N relatie.

Hoe pakken we dit praktisch aan? In eerste instantie maken we een tabel aan ‘Werknemer’. We bepalen een primaire sleutel, in dit geval gebruiken we ‘Personeelsnummer’. In de situatie van een 1-N relatie, dragen we de primaire sleutel van de 1-zijde (coachende werknemer) over als vreemde sleutel naar de N-zijde (gecoachte werknemer). We voegen de primaire sleutel ‘Personeelsnummer’ toe als een bijkomende kolom aan de tabel ‘Werknemer’ waarbij we die een andere naam geven, ‘PersoneelsnummerCoach’ en we duiden die aan als vreemde sleutel met de afkorting ‘FK’ gevolgd door een volgnummer. Dit stelt ons in staat om voor elke werknemer te verbinden met zijn of haar coach omdat elke rij die de werknemer beschrijft een verwijzing bevat.



Figuur 4.7 We zetten unaire relaties om alsof het binaire relaties zijn.

Voor de andere type relaties, loopt de redenering gelijklopend. Mocht een unaire relatie een N-M relatie zijn, maken we gebruik van een tussentabel zoals eerder beschreven werd.

Relaties van een graad 3 en hoger

Een relatie van graad 3 en hoger in een conceptueel datamodel is een situatie waarbij een relatie 3 of meer entiteittypes met elkaar verbindt.

Stel bijvoorbeeld een situatie waarbij een dokter aan een patiënt medicatie voorschrijft. Indien we dit modelleren in een conceptueel datamodel zouden we drie entiteittypes hebben ‘Dokter’, ‘Patiënt’ en ‘Medicatie’ die met elkaar verbonden zijn door een relatie ‘Schrijft voor’. We hebben dit voorbeeld eerder al gebruikt (zie 3.5.5) dus we weten dat we dit niet kunnen herleiden naar binaire relaties.

Om dit vervolgens om te zetten naar een logisch datamodel, creëren we voor elk van de drie entiteittypes een tabel, resulterend in de tabellen ‘Dokter’, ‘Patiënt’ en ‘Medicatie’. Deze drie

tabellen hebben elk een primaire sleutel, respectievelijk 'Dokternummer', 'Patiëntennummer' en 'Medicatiecode'. De relatie vervangen we door een tussentabel 'Voorschrift', waaraan we drie kolommen toevoegen die elk verwijzen naar de primaire sleutels van de tabellen 'Dokter', 'Patiënt' en 'Medicatie'. Elke code vormt een vreemde sleutel in de tabel 'Voorschrift' en duiden we dus aan met de afkorting 'FK' gevolgd door een volgnummer. De drie kolommen samen vormen de primaire sleutel van de tabel 'Voorschrift' en duiden we dus ook aan met de afkorting 'PK'. Voor elke relatie die er bestaat tussen de verschillende entiteiten van de entiteittypes 'Dokter', 'Patiënt' en 'Medicatie' voegen we een rij toe aan de tabel 'Voorschrift'.

Elk attribuut van de relatie, bijvoorbeeld het attribuut 'Datum' om aan te geven wanneer de medicatie werd voorgeschreven, wordt aan de tussentabel 'Voorschrift toegevoegd' als een bijkomende kolom.

We kunnen hetzelfde principe toepassen voor relaties van graad 4, 5, ... Telkens creëren we een tussentabel dat de primaire sleutels bevat van de andere betrokken tabellen.

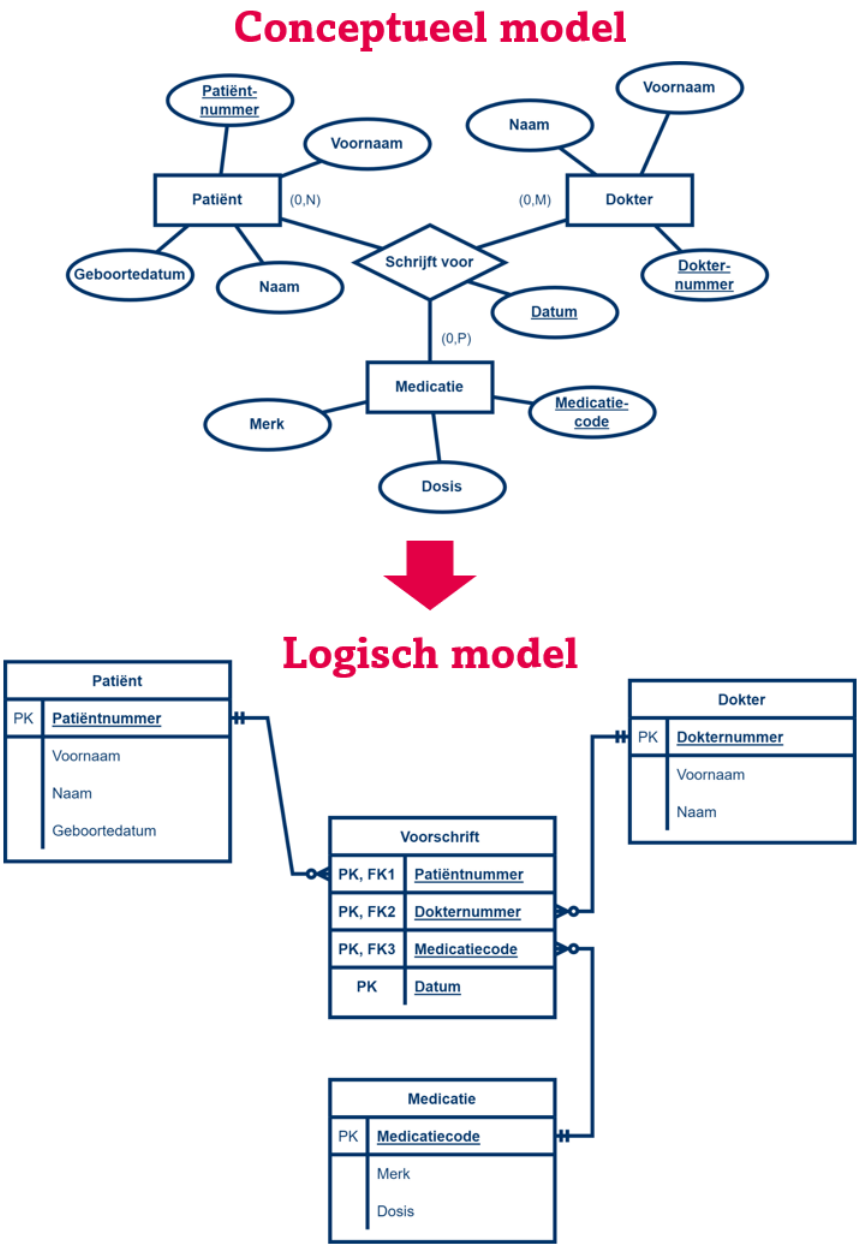
Subtypes en supertypes

We hebben eerder beschreven hoe het ERD uitgebreid werd tot het EERD, waaronder de mogelijkheid om subtypes en supertypes te creëren. Een subtype is een entiteitstype dat attributen en relaties overerft van een supertype en die vervolgens uitbreidt met bijkomende attributen en relaties. Zo zagen we ook dat een supertype vervolgens meerdere subtypes kan hebben, elk een specialisatie van het supertype.

Het voorbeeld dat we eerder gebruikt hebben, was een supertype 'Werknemer', en twee subtypes 'Tijdelijke werknemer' en 'Permanente werknemer'. Van het entiteittype 'Werknemer' houden we de typische eigenschappen bij: naam, voornaam, geboortedatum, werknemersnummer, ... Daarnaast hebben we het entiteittype 'Tijdelijke werknemer' dat alle attributen en relaties overerft van het entiteittype 'Werknemer', maar dit uitbreidt met de attributen 'Startdatum contract' en 'Einddatum contract'. Het entiteittype 'Permanente werknemer' breidt het entiteittype 'Werknemer' uit met de attributen 'Startdatum werknemer' en 'Aantal jaren dienst'.

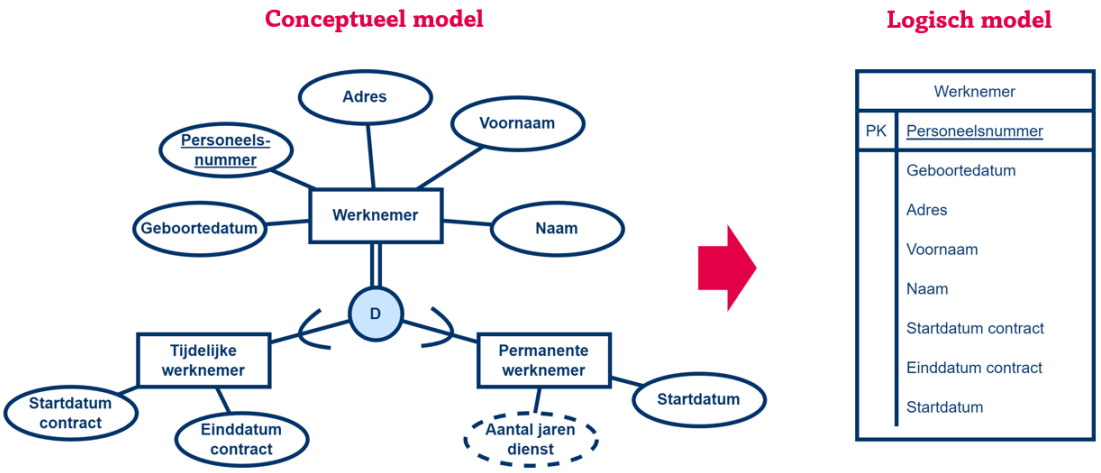
Om dit vervolgens om te zetten naar een logisch datamodel, kunnen we verschillende strategieën toepassen. De afweging tussen de verschillende strategieën gebeurt typisch op basis van performantie, en aangezien we in dit vak nog niet echt de focus leggen op performantie, worden jullie ook niet beoordeeld op jullie keuze. Maar jullie moeten wel in staat zijn om beide strategieën toe te passen.

Een eerste strategie bestaat erin om één tabel te creëren 'Werknemer', waar we voor elk attribuut van het entiteittype 'Werknemer', met uitzondering van de meerwaardige en afgeleide attributen, een kolom toevoegen. We geven deze tabel een primaire sleutel, bijvoorbeeld 'Personeelsnummer'. Vervolgens gaan we voor elk van de attributen van de subtypes van het entiteittype 'Werknemer', met andere woorden de entiteittypes 'Tijdelijke werknemer' en 'Permanent werknemer', ook kolommen toevoegen. Dit resulteert in één grote tabel met



Figuur 4.8 We zetten relaties van graad 3 en hoger om naar een tussentabel.

alle attributen van zowel het supertypes en subtypes. Indien de substypes ook nog relaties hebben, moeten elk van de relaties natuurlijk ook correct verwerkt worden zoals we eerder omschreven hebben. De resulterende tabel bevat één rij voor elke werknemer, de verschillende kolommen worden ingevuld naargelang de werknemer een tijdelijk of permanent werknemer is.

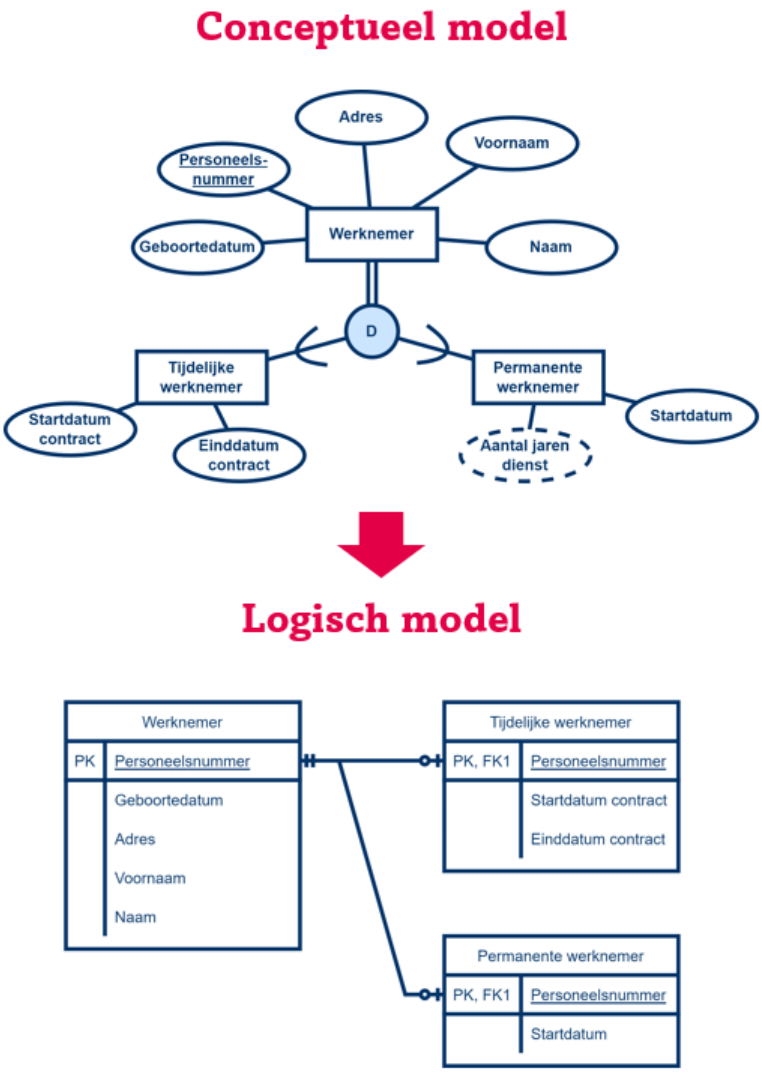


Figuur 4.9 Supertypes en subtypes omzetten naar een logisch datamodel: 1 tabel

Deze strategie werkt zeer goed indien het supertype het grootste aandeel aan attributen en relaties omvat (en de hieruit volgende kolommen), en de subtypes elk slechts enkele bijkomende attributen of relaties hebben. We kiezen dus bewust om geen complexiteit te creëren door bijkomende tabellen bij te maken.

Bijkomend werkt deze strategie goed indien het gaat om overlappende en/of totale overerving. Bij overlappende en/of totale overerving gaan we telkens meer attributen ingevuld hebben van een bepaalde entiteit, of vertaald naar het logisch model zullen er voor een bepaalde rij minder kolommen leeg zijn.

Een tweede strategie bestaat erin om één tabel te creëren 'Werknemer', waar we voor elk attribuut van het entiteittype 'Werknemer', met uitzondering van de meerwaardige en afgeleide attributen, een kolom toevoegen. We geven deze tabel een primaire sleutel, bijvoorbeeld 'Personeelsnummer'. Vervolgens creëren we voor elk van de subtypes bijkomende tabellen, wat voor ons voorbeeld resulteert in de tabellen 'Tijdelijke werknemer' en 'Permanent werknemer'. Ook aan deze tabellen voegen voor elk attribuut van de subtypes, opnieuw met uitzondering van de meerwaardige en afgeleide attributen, een kolom toe. Aan elk van de subtype-tabellen voegen we de primaire sleutel van de supertype-tabel toe. In ons voorbeeld voegen we de 'Personeelsnummer' toe aan de tabellen 'Tijdelijke werknemer' en 'Permanent werknemer'. Gezien het gaat om de primaire sleutel van een andere tabel, duiden we de vreemde sleutel aan met de afkorting 'FK' gevolgd door een volgnummer. Tegelijk gaat deze sleutel ook de rol van primaire sleutel vervullen voor de subtype-tabellen, dus duiden we deze ook aan met de afkorting 'PK'. Dit resulteert in verschillende tabellen, één voor het supertype en één voor elk van de subtypes. Natuurlijk moeten ook hier de verschillende relaties correct verwerkt worden zoals we eerder omschreven hebben. De resulterende supertype-tabel bevat één rij voor elke werknemer. Bijkomend zal er voor elke tijdelijke werknemer nog een rij toegevoegd worden aan de tabel 'Tijdelijke werknemer' en voor elke permanente werknemer een rij aan de tabel 'Permanente werknemer' waarbij telkens de kolommen naargelang worden ingevuld.



Figuur 4.10 Supertypes en subtypes omzetten naar een logisch datamodel: meerdere tabellen

Deze strategie werkt zeer goed indien het supertype een klein aantal attributen en relaties omvat (en de hieruit volgende kolommen), en de subtypes zelf heel wat attributen of relaties hebben. Om de complexiteit van al die bijkomende kolommen die meestal leeg zouden zijn, te vermijden. Splitsen we het geheel dus op.

Bijkomend werkt deze strategie goed indien het gaat om disjuncte en/of optionele overerving. Bij zowel disjuncte en/of optionele overerving hebben we een groot risico op heel wat lege kolommen voor een bepaalde rij, wat dan weer onnodige ruimte inneemt.

Tenslotte kan je bovenstaande strategieën gaan combineren, waarbij je voor bepaalde subtypes een bijkomende tabel creëert, en andere subtypes mee opneemt in de super-type tabel.

4.5 Datamodel normaliseren

Tenslotte willen we nog een kort woordje meegeven rond het concept normalisatie. Normalisatie is het proces waarbij we een datamodel (en de uiteindelijke databank) gaan structureren in logische eenheden. Om ons hierin te begeleiden maken we gebruik van **normaalvormen**, waarbij elke normaalvorm bepaalde voorwaarden oplegt aan de tabellen in het datamodel. De voorwaarden die vervat zitten in deze normaalvormen hebben als doel redundantie te beperken en integriteit te maximaliseren.

De theoretische achtergrond van normaalvormen is gebaseerd op eerste-orde logica waarbij de afhankelijkheden die bestaan tussen de verschillende kolommen, waarbij bijvoorbeeld de kolom 'Postcode' de kolom 'Gemeente' bepaalt, worden weggewerkt om redundantie van data weg te werken.

We verwachten niet dat jullie een model in een bepaalde normaalvorm kunnen omzetten. Wel verwachten we dat jullie het concept normaliseren kunnen uitleggen, en dat jullie bewust omgaan met het creëren van redundantie.

5 Fysiek datamodel

5.1 Doel

In het volgende hoofdstuk maken we de laatste stap bij het plannen van onze databank, het creëren van ons fysiek datamodel We bekijken:

- omzetten van een logisch datamodel naar een fysiek datamodel
- Jullie moeten de volgende termen kunnen uitleggen, hoe we dit weergeven (waar relevant) en toelichten aan de hand van een voorbeeld:
  - Fysiek datamodel
  - Datatype
  - Null-waarde

5.2 Inleiding

De laatste stap in het uittekenen van ons datamodel, is de fysieke laag. Wanneer we aan deze stap komen, hebben we al een databankmodel gekozen en onze conceptueel datamodel omgezet naar een structuur die voldoet aan dit databankmodel, resulterend in het logisch datamodel.

De fysieke laag van ons datamodel beantwoordt de vraag 'Hoe moeten we de datastructuur beschrijven volgens ons gekozen DBMS?'. Daarbij vullen we ons datamodel aan met de volgende informatie:

- het datatype van een kolom
- de lengte van een waarde
- het al dan niet leeg (null) zijn van een waarde
- beperkingen (constraints) die we aan een bepaalde tabel willen opleggen (wordt niet besproken in dit vak)
- indexing (wordt niet besproken in dit vak)

- views (wordt niet besproken in dit vak)
- procedures (wordt niet besproken in dit vak)
- gebruik van diskspace (wordt niet besproken in dit vak)

De reden waarom we deze informatie toevoegen in een aparte laag, is omdat het gaat om de meer technische aspecten van het datamodel. Voor hetzelfde databankmodel, kunnen er verschillen bestaan op niveau van het DBMS. Met andere woorden, een fysiek datamodel voor het ene DBMS kan verschillen van dat van een ander DBMS systeem. De verschillen zijn veelal beperkt, zeker voor DBMS voor databanken volgens het relationeel databankmodel, omdat er standaarden werden vastgelegd. Voor het relationeel databankmodel werden standaarden samengevat in de ANSI standaard voor SQL, voor het eerst in 1986 en meer recent in 2016. De ontwikkelaars van DBMS voor relationele databanken houden zich meestal aan deze standaarden, al voegen ze ook eigen varianten toe om zich te onderscheiden van andere producten op de markt door hun klanten net iets meer aan te bieden. De focus in dit vak ligt bij PostgreSQL, we maken ook geen vergelijking met andere systemen binnen dit vak.

Tenslotte kan een database-architect ook beslissen de data net iets anders te structureren (bijvoorbeeld denormaliseren) dan in het logisch datamodel werd aangegeven. De architect zal dit doen om de performantie of bruikbaarheid van de databank te optimaliseren, vaak met het in overweging nemen van de technische mogelijkheden van het gekozen DBMS. Deze overwegingen komen in een later vak aan bod.

5.3 Van logisch naar fysiek datamodel

Bij de vertaling van een logisch datamodel naar een fysiek datamodel, gaan we in het kader van dit vak de volgende informatie toevoegen:

- we geven de verschillende namen van tabellen en kolommen in lijn met onze afgesproken naamgevingsconventies (naming conventions), daarvoor moeten we eerst naamgevingsconventies vastleggen;
- we bepalen voor elk van de kolommen een datatype en lengte;
- we geven voor elk van de kolommen aan of de waarde in de kolom al dan niet leeg (NULL) kan zijn;
- we leggen beperkingen (constraints) aan een tabel op.

5.4 Naamgevingsconventies (Naming conventions)

In het fysiek datamodel speelt naamgeving (naming conventions) van tabellen en kolommen een grote rol. Je wil vooral naar een naamgeving streven die leesbaar, duidelijk en consistent is. Daarbij laten heel wat DBMS slechts een beperkt aantal karakters toe in de naam van tabellen en kolommen. Je wordt dus vaak gedwongen om het kort te houden.

Er zijn verschillende manieren om regels te bepalen voor naamgeving, en elk bedrijf zal zijn eigen regels gebruiken. Enkele ideeën:

- case van de namen:
  - UPPERCASE
  - lowercase
  - camelCase - de naam start met een lowercase letter, maar elk nieuw woord start met een uppercase letter
  - UpperCamelCase - de naam start met een uppercase letter, en elk nieuw woord start met een uppercase letter
- gebruik van meervoud of enkelvoud
- woorden scheiden van elkaar:
  - met behulp case (zie bovenaan)
  - underscore _
- taal (in een internationale omgeving werk je best altijd in het Engels)
- gebruik van afkortingen om bijvoorbeeld tabellen, primaire of vreemde sleutels aan te duiden. Je kan ook afkorting aanduiden om bepaalde domeinen binnen het datamodel af te bakenen.
- regels om lange woorden af te korten door bijvoorbeeld alle klinkers weg te laten, tenzij het woord start met een klinker.

Daarnaast kunnen we zeker al de volgende tips meegeven:

- gebruik geen spaces
- gebruik geen vreemde karakters
- gebruik geen SQL of DBMS gereserveerde key-words

Zelf passen we de volgende naamgevingsconventies toe:

- **tabel**: we gebruiken een naam in enkelvoud die in lowercase wordt gezet. Verschillende woorden worden van elkaar gescheiden door een '_'.

- **kolom**: de kolomnaam wordt in lowercase beschreven, met verschillende woorden van elkaar gescheiden door een '_'. Probeer de kolomnaam zo kort mogelijk te houden, maar tegelijk wel zo dat het nog mogelijk is om te achterhalen wat het betekent.
- **primaire sleutel**: indien mogelijk, geven we de naam van de sleutel 'id'. Zo is het altijd makkelijk voor de gebruikers om de primaire sleutel te identificeren.
- **vreemde sleutel**: vreemde sleutels zijn verwijzingen naar andere tabellen, dus trachten we de naam van de oorspronkelijke tabel mee in de benaming op te nemen. Bijvoorbeeld de primaire sleutel 'id' uit de tabel 'lector' zou in andere tabellen als vreemde sleutel 'lector_id' genoemd kunnen worden.

Zoals gezegd, je kan zelf een naamgeving van tabellen bepalen, en in jouw toekomstige werkomgeving zal dit deels al gedaan zijn.

5.5 Datatype

In het fysiek datamodel gaan we voor elke kolom het datatype vastleggen. We houden hier rekening met de aard van de data die we in deze kolom willen toevoegen. In de eerste stappen die jullie gezet hebben met SQL, binnen de bundel 'Intro SQL', zijn jullie al in aanraking gekomen met de meest gebruikte datatypes: char(), varchar(), integer en date.

Naast deze datatypes biedt PostgreSQL nog een brede waaier aan bijkomende datatypes. Elk datatype heeft een bepaalde rol en biedt via het gebruik van functies ook not eens bijkomende mogelijkheden.

Bij het bepalen van het datatype moet je dan ook bij sommige kolommen de lengte aangeven. Datatypes kunnen een vaste lengte hebben (bijvoorbeeld integer - 4 bytes, boolean - 1 bit, real - 4 bytes, ...) of een variabele lengte hebben (bijvoorbeeld char, varchar, numeric, ...). Wanneer je de lengte van een kolom aangeeft, moet je rekening houden met de maximale lengte dat een waarde zou kunnen hebben. Een datatype met een variabele lengte laat toe om voor bepaalde kolommen toch heel wat ruimte beschikbaar te stellen, zonder dat die ruimte ook telkens volledig ingenomen wordt. Een goede richtlijn in om dit realistisch te houden en

5.6 NULL of NOT NULL

Bij het fysiek ontwerp moeten we aangeven of de waarde in een kolom al dan niet leeg kan zijn. Voor de kolommen die nooit leeg mogen zijn, kunnen we 'NOT NULL' specificeren. Voor de tabellen waar we dit niet hebben aangegeven, kan de waarde dus wel leeg zijn.

De waarde wat we hier moeten invullen op basis van het logisch datamodel, wordt door twee elementen bepaald. Als eerste element hebben we de vreemde sleutels. Afhankelijk van de relatie die er bestaat tussen twee tabellen door middel van een primaire en een vreemde

sleutel, moeten we op basis van de minimumkardinaliteit evalueren of de waarde NULL kan zijn. Indien de relatie afgedwongen wordt, zoals bijvoorbeeld bij het gebruik van een tussentabel, moeten we aangeven dat de vreemde sleutel 'NOT NULL' is.

Als tweede element moeten we voor elk van de kolommen, buiten de primaire en vreemde sleutels, evalueren of het vanuit de logica van de klant toegelaten is dat een kolom leeg is. Bijvoorbeeld, de naam en voornaam van een werknemer worden niet gebruikt als sleutel, aangezien deze niet noodzakelijk uniek zijn. Maar een werknemer waarvan voor- en achternaam niet ingevuld zijn, is mogelijk niet erg bruikbaar. Vandaar kunnen we bijkomend aangeven dat deze niet leeg mogen zijn.

De waarde NULL kan twee dingen betekenen: enerzijds kan het betekenen dat er een waarde ingevuld zou kunnen zijn, maar dat we deze niet kennen. De waarde zou dan op een later tijdstip ingevuld kunnen worden. Anderzijds zou NULL kunnen aangeven dat voor dit specifiek geval er geen waarde mogelijk is, de kolom is niet van toepassing. Een vergelijking met een NULL waarde geeft zelf ook een NULL waarde, omdat een waarde vergelijken met iets onbekend, is zelf ook onbekend.