Dankwoord

Vele collega's hebben bijgedragen aan de opbouw van deze syllabus. De basis werd geschreven door collega Annick Sennesael. Collega's Joeri Van Steen, Jens Buysse en Joeri Van Herreweghe willen we ook danken voor elke constructieve bijdrage. De talrijke andere lectoren Databanken I danken we voor hun revisies.
Op die manier is deze syllabus het resultaat van de samenwerking van alle betrokkenen.

A. Bernard
A. Van Achter

Gent, september 2018

Voorwoord

Het bewaren en beschikbaar houden van gegevens is een belangrijk aspect van bijna alle automatiseringsprojecten. Hierdoor zijn door alle hypes heen databanken blijvend actueel.

De informaticus moet daarom:

- in staat zijn om een architectuur op te zetten voor het opslaan en het beheren van de gegevens binnen een informatieseringsproject;
- een duidelijk beeld hebben van wat er beschikbaar is op het domein van databanken en wat het meest geschikt is in bepaalde omstandigheden;
- een datamodel kunnen opstellen en omzetten in een operationele databank;
- deze databank kunnen beheren en gebruiken.

De cursus behandelt de voor de praktijk zo belangrijke relationele databanken. Theorie en praktische oefeningen met de relationele databasetaal SQL wisselen elkaar af. De vele voorbeelddatabanken uit de cursus worden ter beschikking gesteld, zodat je er uitgebreid mee kan oefenen.

Inhoudsopgave

Deel I

Relationeel model

Te verwerven competenties

- Kan de verschillende types databanken toelichten en kan die situeren tegenover de 'klassieke' bestanden.
- Kan een (E)ERD opstellen door relevante entiteitstypes, attribuuttypes, de relaties tussen de entiteitstypes en de cardinaliteiten van de relaties af te leiden uit ongestructureerde informatiebron(nen).
- Kan een bestaand (E)ERD evalueren op juistheid en aanpassen op basis van bijkomende of gewijzigde informatie.
- Kan een conceptueel model ((E)ERD) omzetten naar een relationeel datamodel.
- Kan een relationeel datamodel maken met behulp van de normalisatietechniek.
- Kan een relationeel datamodel evalueren op 1NV, 2NV of 3NV.

HOOFDSTUK 1

Databanken gekaderd

1.1 Enkele definities

Database (databank):

- Verzameling van gegevens met een welbepaalde betekenis voor een welbepaalde groep van gebruikers.
- Een databank bestaat uit verzamelingen van persistente gegevens die gebruikt worden door de softwareapplicaties van een bedrijf en beheerd worden door een DBMS (J. Date).
- Een gedeelde verzameling van logisch met elkaar verbonden gegevens en hun beschrijving, ontworpen om aan de informatienoden van een organisatie te voldoen (O. Connolly).

Databasemanagementsysteem (DBMS): Verzameling van programmas waarmee een databank kan worden gecreëerd en beheerd en waarmee gegevens in de databank kunnen worden geladen, gewijzigd en opgevraagd.

= interface tussen gebruikers(programma's) en de databank.

Enkele voorbeelden:

- Oracle
- Microsoft SQLServer
- IBM DB2
- MySQL

- PostgreSql

Databaseprogramma : Computerprogramma dat gegevens uitwisselt met de databank.

1.2 Toepassingen

Databanken zijn niet meer uit ons dagelijks leven weg te denken. Wat je tegenwoordig ook doet, ergens kom je rechtstreeks of onrechtstreeks in aanraking met een databank.

- **Betalen aan de kassa:** Producten worden ingescand, de prijs is automatisch gekend en de voorraad wordt aangepast:
  ⇒deze informatie komt uit een databank
- **Bibliotheek:** Opvragen of een bepaald boek aanwezig is:
  ⇒ informatie van alle uitleningen en boeken zit in een databank
- **Online een vlucht boeken:** Opvragen beschikbaarheid vlucht:
  ⇒informatie van alle vluchten en boekingen zit in een databank
- **Bankautomaat:** Opvragen saldo van je rekening aan de bankautomaat:
  ⇒informatie over de stand van je rekening zit in een databank

1.3 Gegevensmanagement

Gegevensmanagement is het verzamelen, opslaan en terugvinden van gegevens. Maar gegevens op zich zeggen niets. Pas als je er een betekenis aan geeft, wordt het informatie. Met die informatie kan kennis worden gedistilleerd, die een persoon (of een machine) in staat stelt om beslissingen te nemen. Het doel van gegevensmanagement is het verwerven van **informatie** en het ontwikkelen van **kennis**. Hieronder bespreken we de evolutie van het gegevensmanagement.

1.3.1 Gegevensmanagement bij stand-alone applicaties

Bij stand-alone applicaties worden de gegevens beheerd op elk individueel toestel; vaak door elke applicatie afzonderlijk.

Enkele nadelen van deze werkwijze:

- **redundantie:** zelfde gegevens worden meerdere keren in verschillende (types) bestanden bijgehouden en dit op meerdere toestellen;
- **incompatibiliteit:** door structuurverschillen tussen de bestandtypes zijn de gegevens onverenigbaar;

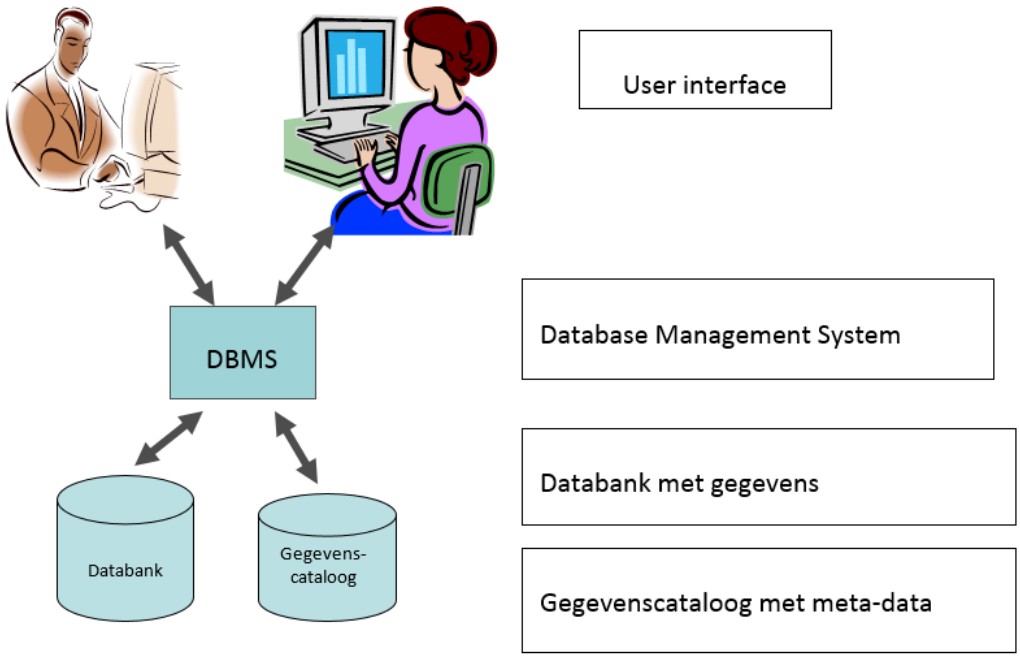
- **inconsistentie:** doordat dezelfde gegevens meerdere keren zijn opgeslagen is de kans reëel dat een wijziging op één plaats niet overal wordt doorgevoerd;
- **inefficiëntie:** een wijziging in de structuur van één bestand kan inhouden dat alle programma's, die dat bestand gebruiken, moeten aangepast worden;
- **weinig flexibiliteit:** indien een bepaalde gebruiker de data op een andere manier wil benaderen (bvb. via postcode de klanten opvragen in plaats van via familienaam) dan moet er hiervoor soms een extra programma geschreven worden.

1.3.2 Gegevensmanagement bij (gedeelde) databanktoegang

Gebruikers of programma's hebben geen rechtstreekse toegang tot de gegevens in de databank. Hun vraag naar gegevens wordt doorgegeven aan het DBMS.

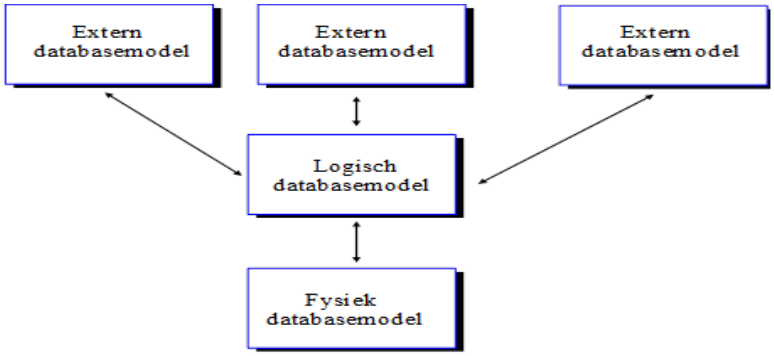
Het DBMS controleert of de aanvrager de juiste bevoegdheden heeft en of de gevraagde data aanwezig is. Zo ja, dan geeft het DBMS de gevraagde gegevens weer.

Onderdelen van een DBMS



Figuur 1.1: Onderdelen van een DBMS

- **User interface:** applicatie om gegevens uit te wisselen met de databank.
- **DBMS:** biedt interfaces aan voor:
  - het definiëren van gegevens
  - het manipuleren van gegevens
  - het bewaken van integriteit
  - het beveiligen: back-up en recovery
  - het beheren: statistieken over efficiëntie en effectiviteit
- **Databank:** bevat de gegevens
- **Gegevenscataloog:** bevat de definities uit de modellen van het 3-lagen gegevensmodel (zie figuur 1.2):



Figuur 1.2: 3-lagen gegevensmodel: een wijziging in één laag mag geen gevolgen hebben voor de andere lagen

  - **fysiek model:** beschrijving hoe de gegevens fysiek zijn opgeslagen.
  - **logisch model:** datadefinities (beschrijving) van alle gegevens in de databank:
    - welke objecten;
    - verbanden tussen de objecten;
    - integriteitsregels.

    Hierin komen evenwel geen implementatie- of opslagdetails in voor!
  - **externe modellen:** view van gebruikers op de gegevens. Dit zijn deelverzamelingen van het logisch model voor één gebruikersgroep of voor één applicatie.

Voordelen van werken met een (gedeelde) databank t.o.v. stand-alone werken:

- beheren van dataredundantie;
- consistente data;
- gegevensonafhankelijkheid;
- opdrachtonafhankelijkheid;
- flexibiliteit.

Nadelen:

- complex;
- kosten;
- grote gevolgen bij defect.

Voorwaarden voor een goede databank:

Vanuit het perspectief van de gebruikers:

- betrouwbaar: de gegevens moeten juist zijn en up-to-date.
- volledig: de databank moet alle gegevens bevatten die de applicatie nodig heeft.
- efficiënt: een gebruiker moet antwoord krijgen binnen een 'redelijke' termijn.
- verstaanbaar: de gegevens moeten voldoen aan vooraf bepaalde eisen voor verstaanbaarheid. Voor bepaalde specialistische doeleinden kan het zijn dat gebruikers scholing nodig hebben, maar in principe zouden ze zonder scholing in hun eigen taal met de databank moeten kunnen werken.

Vanuit het perspectief van de ontwerpers, bouwers en beheerders:

- testbaar: de applicaties op een databank moeten goed te debuggen zijn, er moeten hulpmiddelen zijn om de software te ontdoen van fouten.
- aanpasbaar: er kunnen altijd wijzigingen in gegevensstructuren en applicaties nodig zijn. Het mag niet onnodig moeilijk zijn deze wijzigingen aan te brengen.

Vanuit het perspectief van de organisatie als geheel:

- apparatuuronafhankelijk: organisaties kopen nieuwe computertypes, of fuseren, met de regelmaat van de klok. Databanken moeten daartegen bestand zijn: het database management systeem moet op allerlei hardware en onder allerlei besturingssystemen kunnen draaien.
- organisatieonafhankelijk: wanneer er fusies of samenwerkingsverbanden ontstaan worden dikwijls gegevens gedeeld tussen de betrokken organisaties. Het is dan erg waardevol wanneer de gegevensdefinities die men hanteert met elkaar overeenstemmen. In veel bedrijfstakken heeft men dan ook zogeheten referentie-informatiemodellen opgesteld. Deze modellen zijn standaard gegevensmodellen voor de desbetreffende bedrijfstak en kunnen door elke organisatie worden gebruikt als basis voor de gegevensmodellen van hun eigen databanken.

Voor een gebruiker moet de databank vandaag goed werken, voor een ontwerper, bouwer of beheerder moet de DB morgen nog goed werken, voor de organisatie moet de DB gedurende enkele jaren goed werken, om de investering terug te verdienen.

IT-functies gelinkt aan databanken

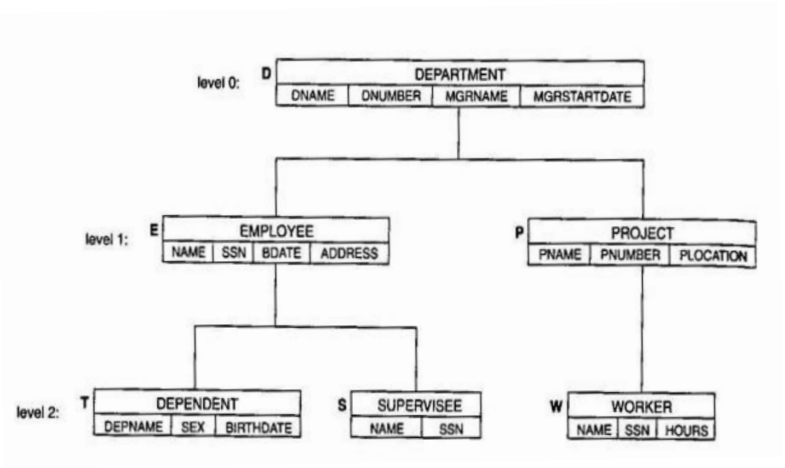
- **DB-ontwerpers:** ontwerpen de databank: maken de modellen;
- **DB-beheerders:** beheren de databank (optimalisatie, statistieken, structuurwijzigingen, ...);
- **DB-gebruikers:** gebruiken de gegevens in de databank (gegevens ingeven, opvragen, wijzigen, verwijderen).

1.4 Geschiedenis

1.4.1 Eerste generatie databankmodellen

Hiërarchische databankmodel

- Ontstaan:
  - Het NASA Apollo maan project (1964) heeft geleid tot de ontwikkeling van GUAM (Generalized Update Access Method) door IBM.
  - Het opzet was kleinere componenten samenvoegen als onderdeel van een groter geheel.
  - Het resultaat werd een omgekeerde boom $\Rightarrow$ hiërarchische structuur. (zie figuur 1.3)
  - In 1966 werd het idee gecommercialiseerd door IBM: IMS (Information Management System).



Figuur 1.3: Voorbeeld van een voorstelling volgens het hiërarchisch model

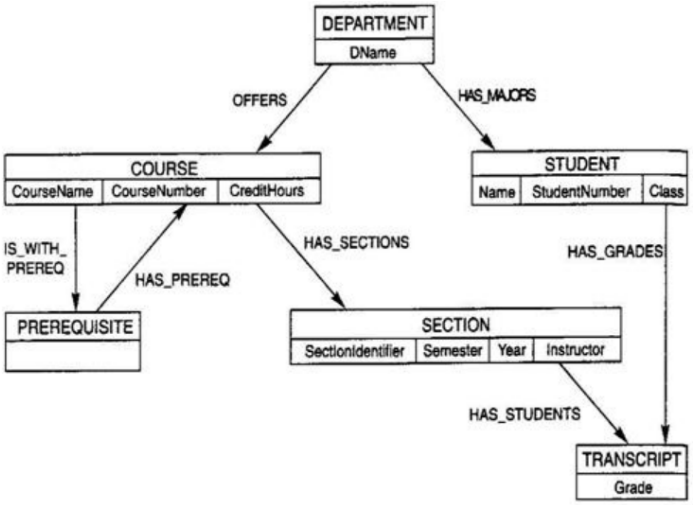
  - IMS is momenteel één van de databankmodellen op mainframes.

- Kenmerken:
  - Elk record in een databank (parent) kan verwijzen naar een n-aantal andere records (children).
  - Elk recordtype heeft één en niet meer dan één eigenaar (owner).
  - Het hiërarchische model kent maar één boomstructuur per databank.
  - De takken hebben zijdelings geen samenhang (alleen parent en children).
  - De enige ingang (root) van de boomstructuur is van bovenaf.
- Nadelen:
  - Er bestaan enkel één-op-veel verbanden.
  - De gegevens zijn niet direct toegankelijk. Navigatie is enkel mogelijk via de parent-child-relaties.

Codasyl: netwerkmodel

- Ontstaan:
  - Ontworpen om nadelen van hiërarchisch systeem weg te werken.
  - Ontwikkeld in 1967 door General Electric: IDS (Integrated Data Store).
  - Is gebaseerd op het netwerk datamodel waardoor complexere dataverbanden kunnen voorgesteld worden.

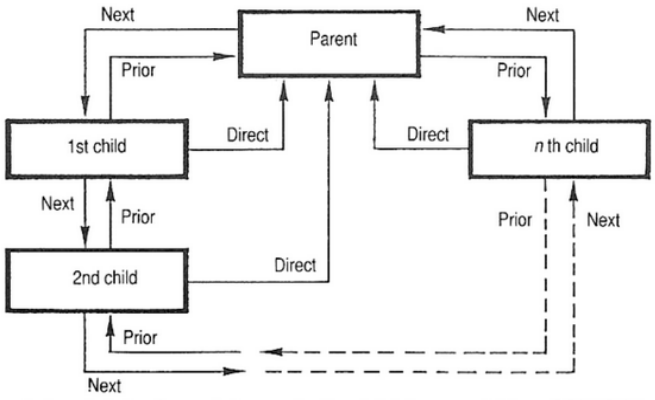
  - In 1969 legt de CODASYL Database Task Group Report de standaarden vast voor netwerkdatabases.



Figuur 1.4: Voorbeeld van een voorstelling volgens het Codasyl- of netwerk-model

- Het model omvat 3 componenten:
  - het netwerk schema: model van de databank gezien door de ogen van de DBA: DBnaam, recordtypes, componenten van elk recordtype;
  - het subschema: het deel van de databank gezien door de ogen van de eindgebruiker of een databaseprogramma;
  - de taal om de data te definiëren en te manipuleren.
- We onderscheiden 3 subtalen:
  - de DDL: Data Definition Language;
  - de Subschema DDL: taal om het deel van de data nodig voor een specifieke applicatie te beschrijven;
  - de DML: Data Manipulation Language
- Voordeel: kan ook veel-op-veel relaties voorstellen
- Nadelen:
  - Het is een heel uitgebreid en bijgevolg 'zwaar' DBMS.
  - Bij elk record moeten wijzers (pointers) opgeslagen worden, die de fysieke representatie van de parent-child-relaties zijn. De pointers verwijzen dus door naar andere records.

– In dit model ontstaat er een probleem wanneer ergens in de databank een pointer corrupt raakt. Hierdoor kunnen hele delen van de databank opeens niet meer bereikbaar zijn.

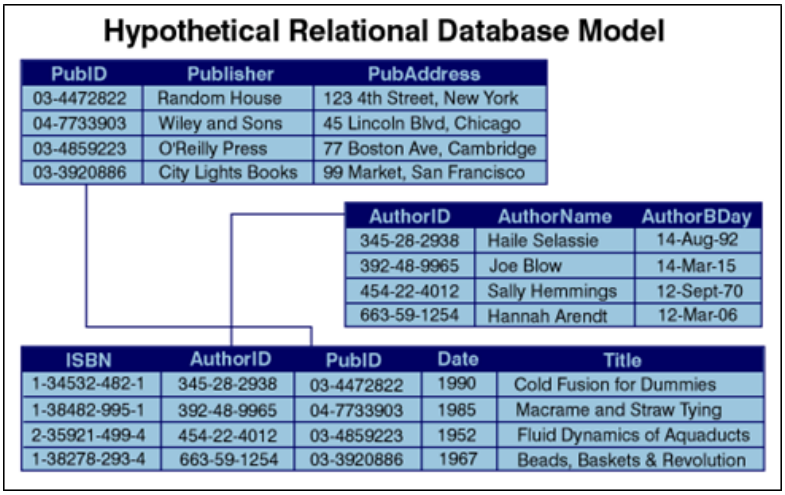


A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

Figuur 1.5: Toegang via pointers

1.4.2 Tweede generatie databankmodellen: Relationele databanken

- Ontstaan:
  - In 1970 schrijft Dr. E.F. Codd, onderzoeker bij IBM, een paper over het relationeel datamodel ('A relational model of data for large shared data banks').
  - Hierna volgt de ontwikkeling van verschillende relationele databanken.
  - In de jaren zeventig werd SQL (Structured Query Language) door IBM ontwikkeld. Pas in de loop van de jaren '80 werd deze taal, gebaseerd op de relationele algebra, de standaardtaal voor relationele databanken.
  - Momenteel bestaan er heel wat commerciële relationele databanksystemen voor alle mogelijke architecturen en besturingssystemen.
  - Enkele voorbeelden: MS SQLServer, MySQL,DB2, Oracle, Ingres, Informix, PostgreSql,
    ldots.
- Kenmerken:
  - wiskundige basis;
  - logische in plaats van fysieke verbanden tussen de gegevens: vreemde sleutels;



Figuur 1.6: Voorbeeld van een relationeel model

- SQL kan onderverdeeld worden in drie subtalen:
  1. DDL: data definition language (DDL)
  2. DML: data manipulation language (DML)
  3. DCL: data control language (DCL).

1.4.3 Derde generatie databankmodellen

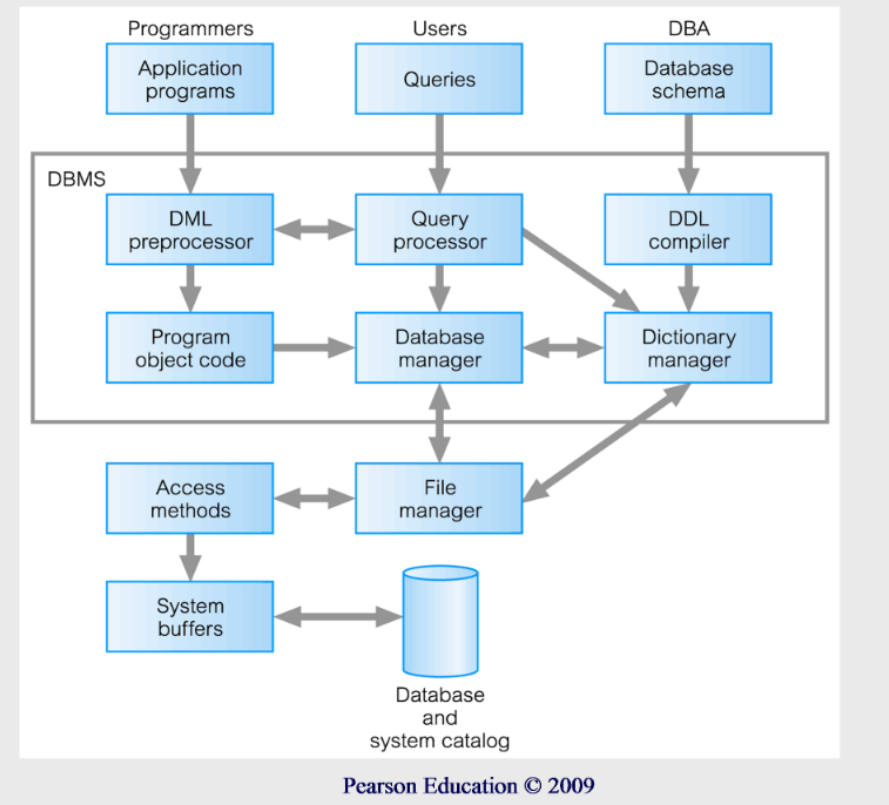
- Object-relationele databanken: relationele DBMSen met objectgeoriënteerde faciliteiten
- Object-georiënteerde databanken (OODBMS)

1.5 Architectuur

1.5.1 Interne architectuur van een DBMS

Een DBMS is samengesteld uit meerdere componenten:

- **DDL compiler**: DDL staat voor Data Definition Language. De DDL compiler ontvangt de definities uit de modellen. Hij zet de DDL-statements om naar een set van tabellen met metadata. Die worden dan weggeschreven naar de gegevenscataloog.
- **Query processor**: vertaalt queries naar een reeks van low-level instructies naar de Database Manager.

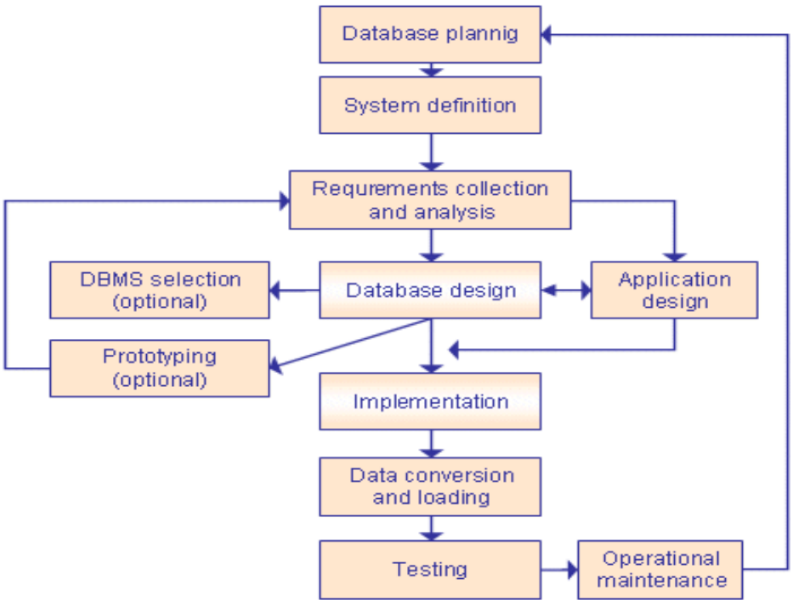


Figuur 1.7: Interne architectuur van een DBMS

- **DML preprocessor**: DML staat voor Data Manipulation Language. De DML preprocessor zet DML-statements (die ingebed zijn in een applicatie) om naar standaard function calls in de hostlanguage. De DML-preprocessor werkt samen met query processor om de code te genereren.
- **Database manager**: ontvangt de queries en onderzoekt de externe en conceptuele schemas om te beslissen welke records nodig zijn om het verzoek uit te voeren. Hierna plaatst de database manager dan een call naar de file manager om de data op te halen.
- **File manager**: zorgt voor de allocatie van opslagruimte op schijf.
- **Dictionary manager**: zorgt voor de toegang naar de systeemcataloog.

1.6 Bouwen van een databank

Het bouwen van een databank gebeurt in verschillende stappen:



Figuur 1.8: Stappenplan voor het bouwen van een databank

1. Database planning: plannen van hoe de verschillende stappen kunnen gerealiseerd worden op de meest efficiënte manier.
2. System definition: afbakenen van de grenzen van het databasesysteem: belangrijkste user views, gebruikers, applicaties.
3. Requirements collection and analysis: verzamelen en analyseren van de vereisten van het te bouwen databasesysteem.
4. Database design:
   a) conceptual database design: bouwen van het conceptueel model
   b) logical database design: bouwen van het logisch model
   c) physical database design: bouwen van het fysieke model
5. Application design: ontwerp van de user interface en de applicatie programmas die gebruik zullen maken van de data in de databank.
6. Implementation: creatie van de fysieke databank en de applicaties.
7. Data conversion and loading: laden van data uit het oude systeem naar het nieuwe; aanpassen bestaande applicaties aan de nieuwe databank.
8. Testing: databasesysteem wordt gecontroleerd op fouten en op vereisten van de eindgebruikers.

9. Operational maintenance: database wordt volledig geïmplementeerd, wordt constant gemonitored en onderhouden. Wanneer nodig zullen nieuwe vereisten geïmplementeerd worden (terug vanaf stap 1).

Deze cursus behandelt:

- Stap 4: in Hoofdstukken 2, 3 en 4 met het Conceptueel model: Entity Relationship Model en Enhanced Entity Relationship Model en in Hoofdstukken 5 en 6 met het Logisch model, het omzetten naar een relationeel model en het normaliseren.
- Stap 5 en 6 komen tijdens de projecten in het tweede semester aan bod.
- Stap 4c, 7, 8 en 9 komen in het 2^{de} jaar aan bod.

HOOFDSTUK 2

Conceptueel model

2.1 Inleiding

Het ontwikkelen van een **conceptueel** gegevensmodel is het in schema brengen van:

- objecten;
- kenmerken van objecten;
- relaties tussen objecten;
- integriteitregels.

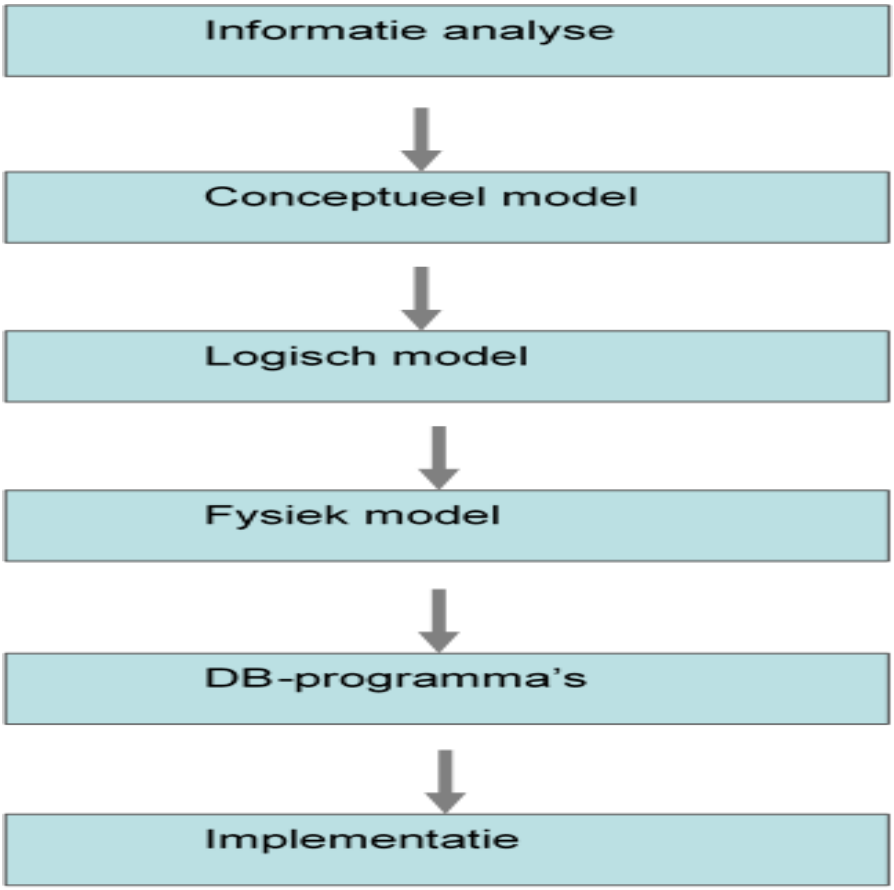
En dit zonder rekening te houden met implementatie: dit model moet kunnen gemapt worden naar zowel een relationele databank, een hiërarchische databank als een object-georiënteerde databank.

Het conceptueel model is dus totaal onafhankelijk van de gebruikte databank, applicaties, programmeertaal, hardware platform of andere fysieke kwesties. Dit model zal als basis dienen voor het opstellen van het logisch model. Aan de hand van het logisch model wordt het fysiek model gemaakt.
DB-programma's worden geschreven (of herschreven) om gegevens uit de databank te halen of om data naar de databank weg te schrijven.
Daarna volgt de implementatie. (zie hoofdstuk 1: bouwen van een databank)

2.2 Entity Relationship Model

Als voorbeeld van een conceptueel model bespreken we het Entity Relationship Model, dat werd ontwikkeld door Chen (1976). Het wordt veel gebruikt omwille van:

- de verstaanbaarheid;



Figuur 2.1: Stappenplan voor de ontwikkeling van een databank

- de grafische formulering (ERD);
- de tools (b.v.b. ERWIN).

Er zijn verschillende varianten op dit model vooral wat betreft de grafische formulering. De gemeenschappelijke elementen zijn:

- de entiteittypen en attribuuttypen;
- de relatietypen.

Bouwstenen

In de werkelijkheid komen allerlei objecten voor waarover wij gegevens willen verzamelen. In het ER-model worden deze objecten **entiteiten** genoemd. De eigenschappen van de objecten zijn **attributen**.

Verbanden tussen objecten worden gelegd aan de hand van **relaties**.

Entiteittype
Dit is een verzameling van gelijksoortige entiteiten.
Een entiteit is iets of iemand die bestaat en die te onderscheiden is. Het heeft dus een inhoud en een identiteit.
Een entiteit kan fysisch bestaan (een klant, een student, een factuur, ...) of is abstract (een beroep, een les, ...).
Een entiteit heeft kenmerken (eigenschappen zoals naam van een klant, adres, telefoonnummer, ...).
We modelleren niet de objecten op zich (Jan, Karin, Chantal, ...) maar wel het **type** (Klant).

Modelleren = nagaan welke objecten (entiteiten) in de databank moeten opgenomen worden (analyse van de informatiebehoefte) en deze in een verstaanbaar model gieten.
We nemen enkel deze entiteittypen op die nodig zijn om ons DOEL te bereiken. We modelleren dus niet de ganse wereld maar enkel die objecten die nodig zijn om het beoogde resultaat te bekomen.

Attribuuttype
Een attribuuttype is een benoemde karakteristiek van een entiteit.
Ook hier nemen we enkel die attribuuttypes op die relevant zijn.
We nemen ook geen procesgegevens op.
Er zijn verschillende soorten van attribuuttypes: een attribuuttype kan enkelvoudig zijn of samengesteld, éénwaardig of meerwaardig, kan een sleutelattribuut zijn of een niet-sleutelattribuut.

Enkelvoudige attribuuttypes: zijn atomair, kunnen niet meer opgesplitst worden.
b.v.b. voornaam van een klant.

Samengestelde attribuuttypes: kunnen nog opgesplitst worden in deelattributen.
b.v.b. naam van een student $\Rightarrow$ voor- en familienaam.

Eénwaardige attribuuttypes: kunnen, binnen de context waarin wordt gewerkt, op elk tijdstip slechts één waarde bevatten.
b.v.b. ISBN van een boek, geboortedatum van een student, verkoopprijs van een brood.

Meerwaardige attribuuttypes: kunnen, binnen de context waarin wordt gewerkt, op elk tijdstip meerdere waarden, bevatten.
b.v.b. auteurs van een boek, hobby's van een student.

Sleutelattribuut: unieke identificatie van een entiteit.
b.v.b. ISBN $\Rightarrow$ identificatie boek.

Kan uit meerdere attribuuttypen bestaan.
b.v.b. jaar + richting + groep ⇒identificatie van klas.

Indien het combineren van attributen nog geen unieke identificatie oplevert, dan spreken we over een zwak entiteittype (zie verder).

Relatietype
Een relatie is een verband tussen 2 of meer individuele entiteiten.
b.v.b. Jan isGetrouwdMet Annick

Een relatietype is de verzameling van gelijksoortige relaties (verbanden).
b.v.b. huwelijk

Een relatietype kan ook atributen hebben.
b.v.b. huwelijksdatum

Een relatietype is van een bepaalde **graad**: het aantal entiteittypes dat deelneemt aan de relatie:

- binair: b.v.b. Student zit in Klas;
- unair: b.v.b. Persoon is getrouwd met Persoon;
- hogere graad: bespreken we niet in deze cursus.

Een relatietype heeft 2 **rollen**: een relatie heeft 2 richtingen:

- klas **bevat** studenten, student **zit** in een klas;
- dokter **behandelt** patiënt, patiënt **wordt behandeld door** dokter;
- unaire relatie tussen 2 personen: ouder en kind;
- unaire relatie tussen 2 personen: rol: echtgenoot.

Een relatietype heeft **constraints**: de beperkingen van een relatie:

Maximumcardinaliteit: het maximum aantal entiteiten dat deelneemt aan de relatie (1 of N):

- 1-op-1 relatie: Een afdeling heeft maximum 1 chef, een chef is maar hoofd van max. 1 afdeling.

- 1-op-veel relatie: Een leerling zit in maximum 1 klas, een klas heeft max. veel leerlingen.
- veel-op-veel relatie: Een docent geeft les aan meerdere studenten, een student krijgt les van meerdere docenten.

Minimumcardinaliteit: verplicht (1) of niet verplicht (0).

B.v.b. In een firma is elke bedrijfswagen toegewezen aan een personeelslid, maar niet elk personeelslid heeft een bedrijfswagen.

Personeelslid *rijdtMet* niet verplicht een Wagen.
Personeelslid *rijdtMet* min 0 Wagen.

Omgekeerde richting:
Wagen *isVan* verplicht 1 Personeelslid.
Wagen *isVan* min 1 Personeelslid.

Een verplichte minimumcardinaliteit wijst op *bestaansafhankelijkheid*. In vorig voorbeeld is Wagen bestaansafhankelijk van Personeelslid.

2.3 ERD (Entity Relationship Diagram)

Dit is de grafische voorstelling van het entity relationship model.

Er zijn verschillende notaties, wij gebruiken de UML-notatie:

- Entiteittype in kader: naam begint met een hoofdletter;
- Attributen: beginnen met een kleine letter;
- Relatietype begint met een kleine letter plus leesrichting ($\rightarrow$ of $\leftarrow$);
- Min. cardinaliteit .. max. cardinaliteit aan beide zijden van het relatietype;
- Alle enkelvoudige kandidaatsleutels worden onderlijnd. Indien een sleutel uit meerdere attribuuttypes bestaat (samengestelde kandidaatsleutel), duiden we dit dan aan met de 'u'-constraint.

Op die manier kan het ERD in figuur 2.2 volgt gelezen worden:
Een klas wordt geïdentificeerd aan de hand van de combinatie van jaar, richting en groep. Een klas kan meerdere studenten hebben. Een student heeft een stamnummer, naam en e-mailadres. Een student kan geïdentificeerd worden ofwel aan de hand van zijn stamnummer ofwel aan de hand van zijn e-mailadres.
Een student zit altijd in (juist) 1 klas.

Figuur 2.2: Entity Relationship Diagram

HOOFDSTUK 3

Conceptueel model: zwakke entiteiten

3.1 Zwak entiteittype

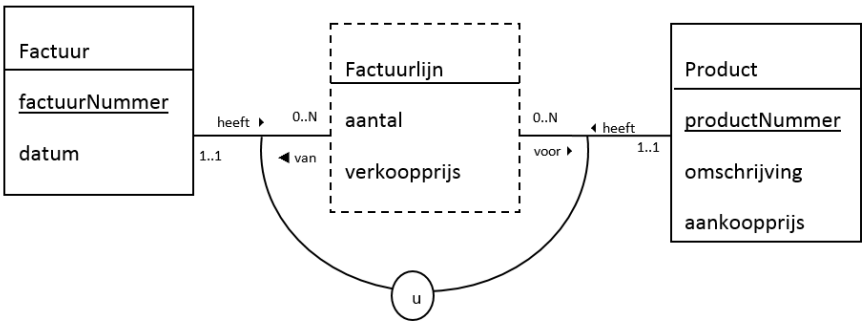
Wanneer een entiteittype op basis van zijn eigen attribuuttypes zich niet kan identificeren, noemen we dit een zwak entiteittype. Een zwak entiteittype kan hierdoor **nooit op zichzelf bestaan** en is dus **altijd bestaansafhankelijk** van andere entiteittypes.

Een zwak entiteittype kan enkel uniek geïdentificeerd worden op basis van de relaties die het heeft met de entiteittypes waarvan het bestaansafhankelijk is. Op basis hiervan zal een uiteindelijke sleutel bepaald worden. Het proces hiervoor zien we in hoofdstuk 3: Mapping.

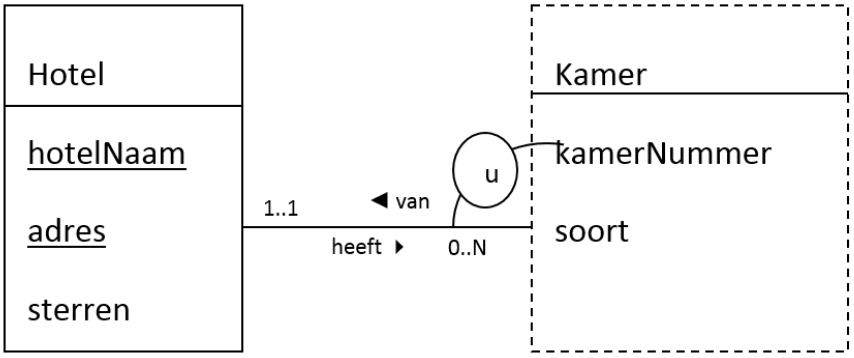
Omgekeerd kan niet gesteld worden dat elk bestaansafhankelijk entiteittype zwak is. Zoals in vorig genoemd voorbeeld is Auto bestaansafhankelijk van Personeelslid. Toch heeft een auto niet de sleutel van een personeelslid nodig om zich te kunnen identificeren. Een auto kan zich identificeren aan de hand van chassisnummer en is dus sterk.

Een zwak entiteittype wordt in stippellijn getekend in een ERD en identificatie zal altijd gebeuren aan de hand van 1 of meerdere relaties die dat zwak entiteittype heeft met andere entiteittypes waarvan het bestaansafhankelijk is (cardinaliteit 1..1).

Factuurlijn (zie figuur 3.1) is zwak ten opzichte van Factuur én Product en kan zich enkel identificeren aan de hand van de combinatie van factuurNummer en product-Nummer. Dit duiden we in een ERD aan met de 'u'-constraint. Het nummer van de kamer (zie figuur 3.2) is niet voldoende om een kamer te identificeren. In de databank zitten immers meerdere hotels. Kamernummer 1 zal dus hoogstwaarschijnlijk meerdere keren voorkomen en telkens een andere kamer beduiden. In combinatie met het hotel kan een kamer wel geïdentificeerd worden.

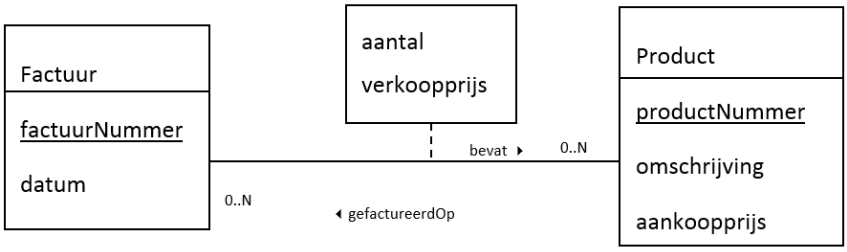


Figuur 3.1: Voorbeeld 1: Zwak entiteittype



Figuur 3.2: Voorbeeld 2: Zwak entiteittype

Opmerking: indien een zwak entiteittype bestaansafhankelijk is van twee entiteittypes, telkens in een 1-op-veel relatie, dan kan in een entity relationship model dit ook voorgesteld worden door een veel-op-veel relatie tussen de 2 sterke entiteiten. De attributen van het zwakke entiteittype worden dan *relatie-attributen*:



Figuur 3.3: Voorbeeld 3: Gebruik maken van relatie-attributen

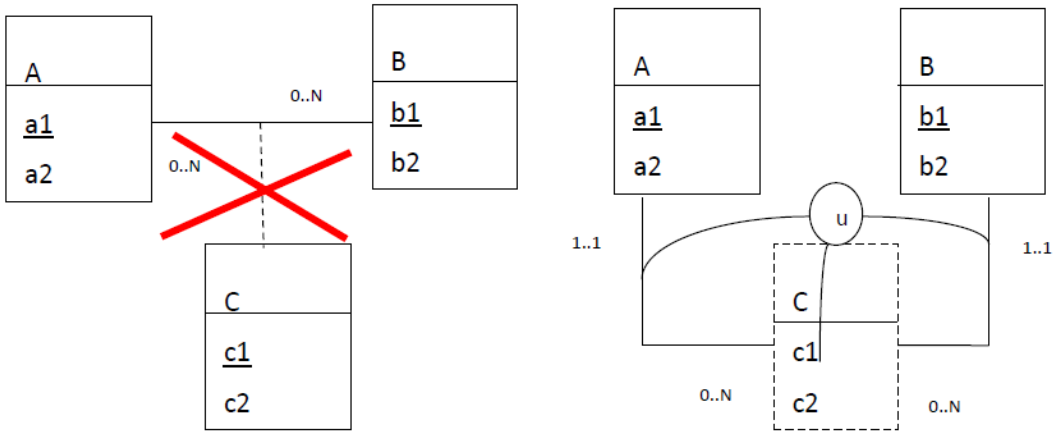
3.2 Relatie-attributen

Zoals in vorig voorbeeld vermeld, kan een relatie ook attributen hebben. Die attributen komen enkel voor, voor die entiteiten die verbonden zijn door de relatie. Dus enkel voor producten die gefactureerd zijn, houden we het aantal en de verkoopprijs bij. Aantal is geen attribuut van *Product*, want de ene keer zal men 2 stuks van het product factureren en op een volgende factuur zijn het er misschien 10. Je kan dus aantal niet als attribuut van *Product* beschouwen. Aantal is ook geen attribuut van *Factuur* want op een factuur staan meerdere producten met elk hun eigen aantal. Het attribuut aantal komt dus enkel voor in de relatie tussen een welbepaald product op een welbepaalde factuur. Hetzelfde geldt voor de verkoopprijs.

In een ERD stelt men relatie-attributen voor in een kader, verbonden met de relatie door een stippellijn.

Opgelet!! Wij zien in de cursus enkel relaties tot de 2^e graad. Teken dus nooit een entiteittype in een relatie tussen 2 andere entitteittypen!!

Indien de relatie-attributen een verband hebben met elkaar en zichzelf kunnen identificeren in de relatie, dan moet je hiervoor een apart (zwak!) entiteittype aanmaken: zie figuur 3.4.

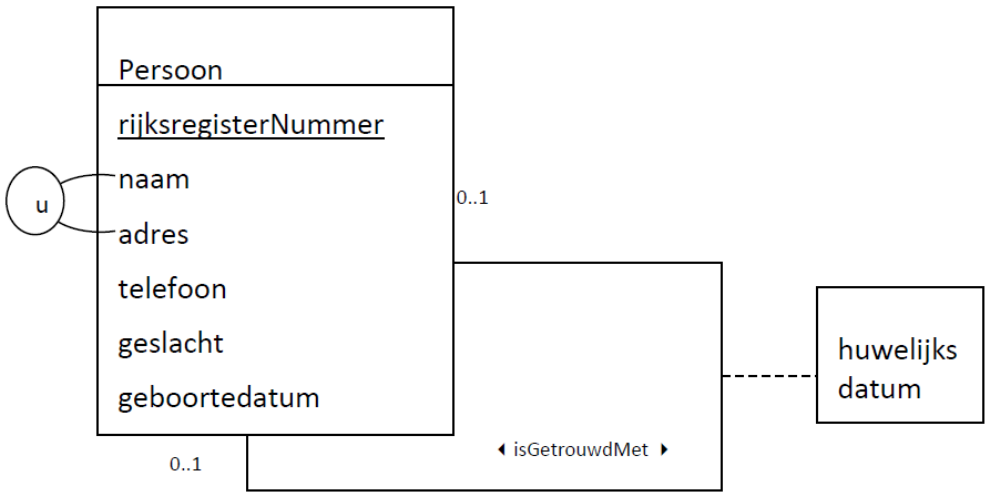


Figuur 3.4: Geen 3^e-graad relaties!

3.3 Historiek

Historische gegevens kunnen voorgesteld worden in een entity relationship model aan de hand van een zwak entiteittype met een datum en eventueel een tijdstip.

Voorbeeld:
Huwelijk tussen 2 personen: een persoon kan op een bepaald ogenblik maar met één andere persoon getrouwd zijn en het is natuurlijk geen verplichting om te trouwen.
We hebben dus een unair 1-op-1 relatietype met als relatie-attribuut 'huwelijksdatum':



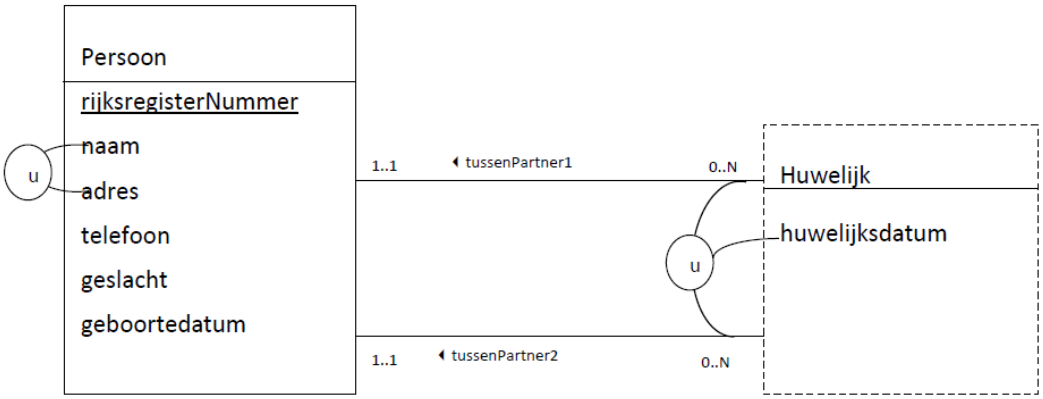
Figuur 3.5: Tijdsaspect in ERD

In dit model (figuur 3.5) is er geen historiek. Indien een koppel beslist om te scheiden, wordt de relatie tussen de 2 entiteiten verwijderd. In de databank is geen spoor meer terug te vinden dat die personen ooit met elkaar getrouwd waren. Wenst men een historiek dan zal het model als volgt wijzigen:

1. een zwak entiteittype Huwelijk met attribuuttype huwelijksdatum wordt toegevoegd;
2. dit entiteittype is verbonden met Persoon door 2 1-op-veel relaties (zie figuur 3.6).

Een huwelijk wordt geïdentificeerd aan de hand van huwelijksdatum + identificatie Partner1 + identificatie Partner2.

Soms is een datum niet genoeg om een historiek voor te stellen. Indien een gebeurtenis



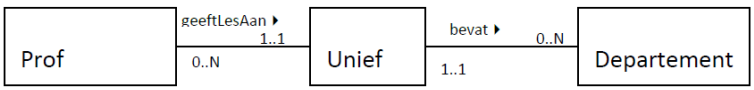
Figuur 3.6: Historiek

meerdere keren op één dag kan plaatsgrijpen zal het ook nodig zijn om een tijdstip op te nemen.

3.4 Mogelijke problemen bij het gebruik van het entity relationshipmodel

Bij het opmaken van een ERD bestaat de kans op zogenaamde *connection traps*. Een connection trap is aanwezig wanneer het onmogelijk is om de benodigde informatie éénduidig op te halen doordat er verkeerde relaties liggen tussen de entiteiten, of doordat er relaties ontbreken. Er zijn 2 soorten connection traps:

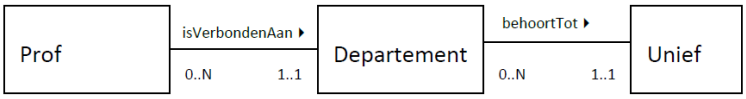
- **Fan trap**: wanneer 2 "1-op-veel" relaties verbonden zijn met hetzelfde entiteittype. Voorbeeld: Aan welk departement is de professor verbonden? Deze vraag kan in



Figuur 3.7: Fan trap

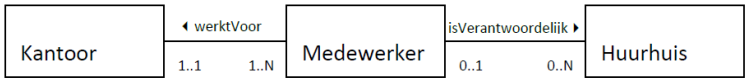
  het ERD van figuur 3.7 niet beantwoord worden. Om op deze vraag een antwoord te kunnen formuleren moet het ERD aangepast worden zoals bijvoorbeeld in figuur 3.8.

- **Chasm trap**: wanneer een model suggereert dat er een relatie is die voor sommige entiteiten niet geldt. Meestal het gevolg van een minimumcardinaliteit 0.



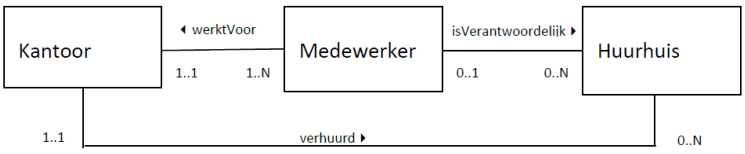
Figuur 3.8: Fan trap: oplossing

Voorbeeld: Stel in figuur 3.9 dat een huurder geïnteresseerd is in een bepaald huis.



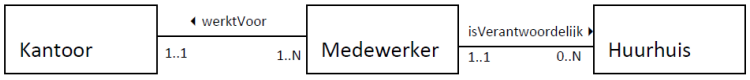
Figuur 3.9: Chasm trap

In welk kantoor kan hij terecht? Oplossing: een extra relatie tussen Kantoor en Huurhuis (figuur 3.10).



Figuur 3.10: Oplossing via een extra relatie

Een andere oplossing kan zijn het verplicht maken van de relatie tussen Huurhuis en Medewerker (figuur 3.11).
Opgelet: hierdoor gaan we wel de eis opleggen dat een huurhuis steeds aan een medewerker gekoppeld moet worden, terwijl dit misschien niet altijd zo is. In dit geval moet je de opdrachtgever vragen of dat dit gewenst is.



Figuur 3.11: Oplossing via aanpassing minimumcardinaliteit

HOOFDSTUK 4

Conceptueel model: Enhanced Entity Relationship Model

4.1 Definitie

Het Enhanced Entity Relationship Model (EER-model) is een uitbreiding van het klassieke ER-model. Het gebruikt de bouwstenen van het ER-model en voegt een aantal abstracties toe zodat meer betekenisaspecten van gegevens kunnen worden gemodelleerd. De belangrijkste uitbreidingen zijn specialisatie/generalisatie, aggregatie en compositie.

4.2 Specialisatie

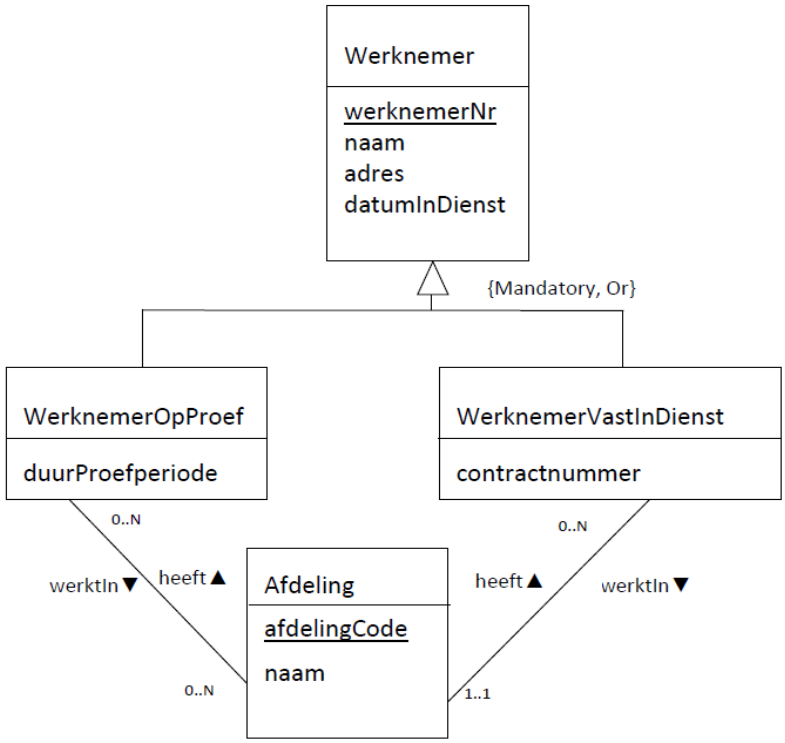
Specialisatie kan men toepassen indien men vaststelt dat sommige entiteiten in een entiteittype naast gemeenschappelijke kenmerken, ook specifieke kenmerken hebben. De algemene kenmerken worden dan verzameld in een *supertype*, de specifieke in *subtypes*.

Een ***subtype*** is een deelverzameling van entiteiten van een entiteittype die zich op een bijzondere wijze onderscheidt en waarvan het zinvol is om ze te onderscheiden omdat gebruikers in de betekenis van zo een groep geïnteresseerd zijn.

Het ***supertype*** bevat de algemene kenmerken, de subtypes de specifieke. De subtypes erven de eigenschappen van het supertype.

Een specialisatie kan altijd gelezen worden als 'IS EEN', terwijl een relatie gelezen wordt als 'HEEFT'.

Stel dat een werknemer altijd verbonden is aan één welbepaalde afdeling in het be-



Figuur 4.1: Voorbeeld specialisatie

drijf, behalve als hij in zijn proefperiode is, dan werkt hij in meerdere afdelingen om het bedrijf beter te leren kennen. Dit kon niet voorgesteld worden in het ER-model. Met specialisatie kunnen we dit wel oplossen. Een werknemer is nu onderverdeeld in 2 subcategorieën: hij is ofwel op proef of is vast in dienst. Voor elke 'soort' werknemer kunnen nu de juiste cardinaliteiten vastgelegd worden.

4.2.1 Participatie constraint: Optional of Mandatory

Optional

Een specialisatie is optioneel als niet elke entiteit uit het supertype tot een subtype behoort.

Bijvoorbeeld: supertype Zoogdier, subtypes Mens en Aap. Er zijn nog andere zoogdieren in de databank dan apen en mensen, maar we hebben geen behoefte om daar specifieke eigenschappen van op te nemen.

Mandatory

Een specialisatie is verplicht als elke entiteit uit het supertype tot een subtype behoort.

Bijvoorbeeld: supertype Persoon, subtypes Werknemer, Klant, Leverancier. In de databank nemen we dan enkel personen op die werknemer, klant of leverancier zijn.

4.2.2 Disjoint constraint: And of Or

Or

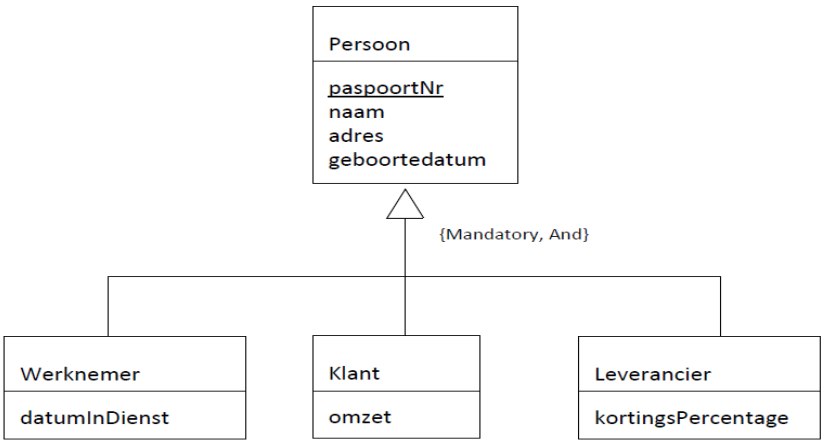
Disjoint: een entiteit kan maar tot één subtype behoren (exclusieve OR).

Bijvoorbeeld een zoogdier kan niet tegelijk een aap en een mens zijn.

And

Nondisjoint: een entiteit kan tot meerdere subtypes behoren.

Bijvoorbeeld een persoon kan zowel werknemer als klant zijn in de databank van een bedrijf.



Figuur 4.2: Voorbeeld Mandatory / And

4.3 Aggregatie/Compositie

Aggregatie stelt een 'is onderdeel van'-relatie voor. Eén entiteittype stelt het geheel voor, het andere entiteittype het onderdeel. Aggregatie is enkel een conceptueel begrip om een geheel van een onderdeel te onderscheiden.

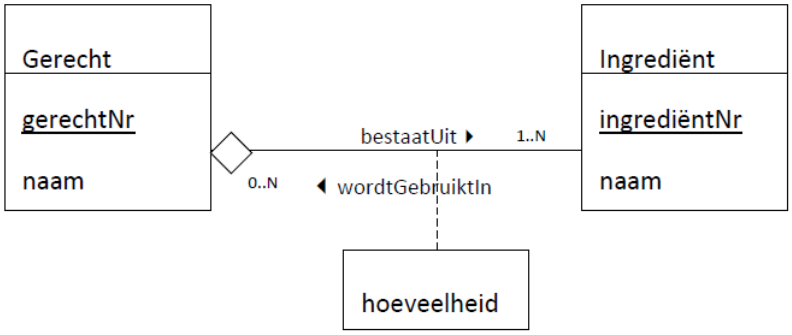
Indien het bestaan van het onderdeel afhangt van het geheel dan spreken we over

compositie. Dus bij compositie zal de entiteit die het onderdeel voorstelt, automatisch verdwijnen als de entiteit die het geheel voorstelt, verwijderd wordt.

Een aggregatie wordt voorgesteld door een open ruit, een compositie door een opgevulde ruit. Een aggregatie kan altijd gelezen worden als BESTAAT UIT.

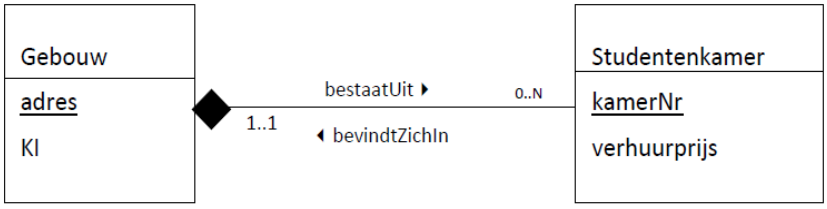
Voorbeeld aggregatie:

Een gerecht uit de databank van een traiteur gebruikt een aantal ingrediënten. Een ingrediënt wordt gebruikt in meerdere gerechten. Een gerecht is het geheel, de ingrediënten de onderdelen. Als een gerecht niet meer gebruikt wordt (verwijderd wordt), kunnen de gebruikte ingrediënten wel nog op zich bestaan. Het is niet omdat de pizza van het menu geschrapt wordt, dat het ingrediënt kaas uit de databank moet verdwijnen. Dit is duidelijk een voorbeeld van aggregatie (figuur 4.3).



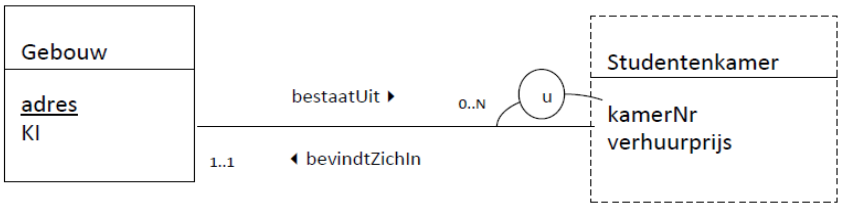
Figuur 4.3: Voorbeeld aggregatie

Voorbeeld compositie:
Een gebouw is onderverdeeld in verschillende studentenkamers. Als het gebouw gesloopt wordt, dan verdwijnen automatisch ook de studentenkamers. De levensduur van een studentenkamer hangt af van de levensduur van het gebouw.



Figuur 4.4: Voorbeeld compositie

Opmerking: een entiteittype dat de rol van 'onderdeel' heeft in een compositie kan ook voorgesteld worden als een zwak entiteittype. In bovenstaand voorbeeld is Studentenkamer sterk. De kamers zullen dus gewoon doorgenummerd worden zonder rekening te houden met het gebouw. Meer waarheidsgetrouw is om van Studentenkamer een zwak entiteittype te maken zodanig dat zijn identificatie afhangt van Gebouw: zie figuur 4.5.



Figuur 4.5: Een compositie via een zwakke entiteit voorstellen

4.4 Besluit

Om de verbanden tussen verschillende entiteittypen voor te stellen, hebben we verschillende instrumenten:

- een relatietype stelt een HEEFT EEN-verband voor;
- een specialisatie stelt een IS EEN-verband voor;
- een aggregatie of compositie stelt een BESTAAT UIT-verband voor.

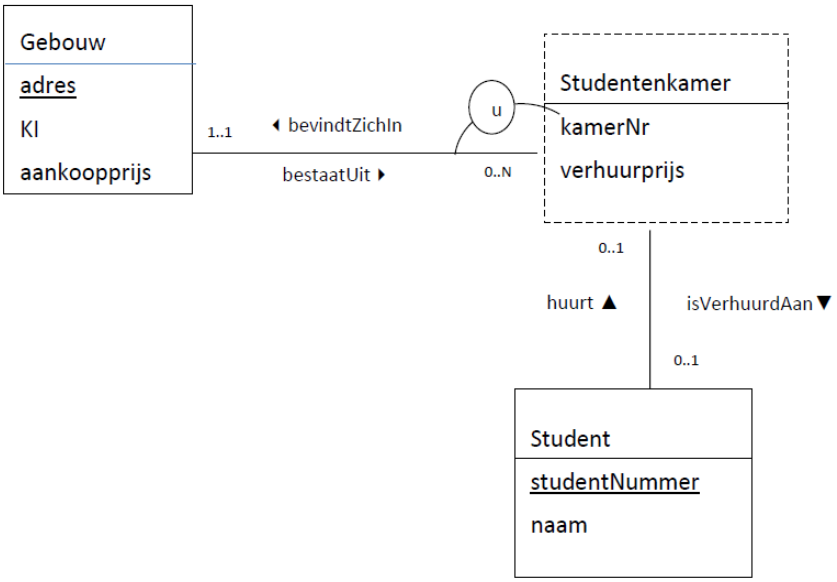
Aggregatie of compositie worden altijd gebruikt in combinatie met een relatietype. Een compositie kan ook voorgesteld worden door van het onderdeel een zwak entiteittype te maken.

Wanneer is je model goed? Je conceptueel model is goed wanneer alle vragen naar informatie van de eindgebruiker kunnen opgelost worden.

Door de use cases te herlezen en door gesprekken met de eindgebruiker kan gecontroleerd worden of alle benodigde data voorzien is in het model.

Een veelgebruikte (en goedkope) methode is: stel aan de hand van het conceptueel model een tekst op en laat die tekst nalezen door de gebruiker.

Een verkeerde structuur zal hierdoor vrij snel aan het licht komen. Vraag ook aan de gebruiker welke taken er gebruik zullen maken van de databank. Met andere woorden op welke vragen de databank een antwoord moet kunnen geven.



Figuur 4.6: Is je model goed?

Voorbeeld: Het model moet in staat zijn om volgende vragen te beantwoorden:

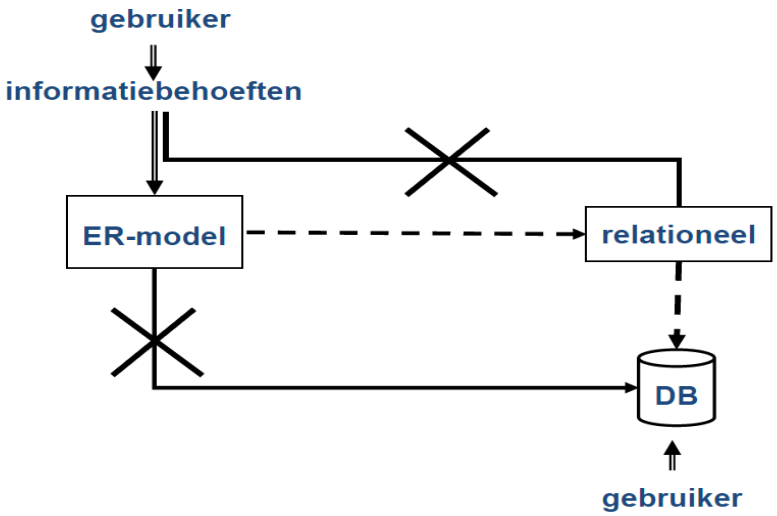
- Aan welke student (naam) is studentenkamer Kerkstraat 10, 2 momenteel verhuurd?
- Wat is de verhuurprijs van studentenkamer Kerkstraat 10, 1?
- Wat is het gsm-nummer van de student die kamer nummer 5 uit de Kerkstraat 12 in Gent huurt?
- Aan welke student was studentenkamer Kerkstraat 12, 3 vorig jaar verhuurd?
- Heeft student nummer 82 (Jan Pieters) deze maand zijn huursom reeds betaald?

Je zal merken dat met bovenstaand model niet alle vragen kunnen opgelost worden. Pas het model aan zodanig dat dit wel het geval is.

HOOFDSTUK 5

Logisch model

5.1 Inleiding



Figuur 5.1: Toegang tot databank

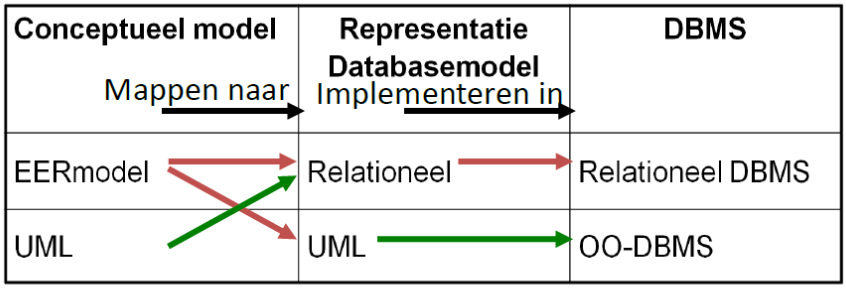
In het vorige hoofdstuk hebben we het Entity Relationship Model als voorbeeld van een conceptueel model bestudeerd. In een conceptueel model wordt geprobeerd om de werkelijkheid waarheidsgetrouw weer te geven. Er wordt (nog) geen rekening gehouden met de mogelijkheden en beperkingen inherent aan de databank die zal gebruikt worden. Het grote voordeel van deze aanpak is dat de resulterende modellen gemakkelijk te verstaan en aan te passen zijn.

Een DBMS kan enkel werken met een databank waarin de gegevens gestockeerd zijn op een manier die het DBMS voorschrijft. De door een DBMS opgelegde typestructuur

wijkt meestal af van de structuur van het conceptuele model. In dit hoofdstuk behandelen we dan ook de vraag: Hoe de gegevens die opgenomen zijn in het conceptuele datamodel te structureren zodanig dat ze opgeslagen kunnen worden in een relationeel databasemanagementsysteem? Het model dat we bekomen na het oplossen van deze vraag, noemen we een *logisch model* of meer specifiek een *relationeel model*.

Concreet betekent dit dat we het conceptuele (E)ER gegevensmodel omzetten naar een relationeel model door de gegevens in het gegevensmodel te herstructureren volgens de principes van het relationeel databasemodel. Een probleem dat hierbij kan ontstaan is dat er informatie over het probleemdomein verloren gaat. We zullen dus rekening moeten houden met de beperkingen van de relationele databasetechnologie. Belangrijk is dat we goed bijhouden welke informatie verloren is gegaan (zodat we hier later oplossingen voor kunnen zoeken).

Waarom niet direct van start gaan volgens de principes van het relationele model? Er zijn goede redenen om dit niet te doen. Zoals je later in dit hoofdstuk zal zien is de semantische uitdrukkingskracht van het relationele model minder dan dat van het ER model. Bij conceptuele modellering is het doel de werkelijkheid zo dicht mogelijk te benaderen, terwijl het relationele model als doel heeft om de type-structuur van de relationele databank zo dicht mogelijk te benaderen. We opteren dus om eerst een conceptueel model op te stellen dat de werkelijkheid van de gebruiker(s) zo dicht mogelijk benadert. Nadien 'mappen' we dit conceptueel model naar een logisch model dat kan geïmplementeerd worden in een DBMS.



Figuur 5.2: Conceptueel model mappen tot relationeel model

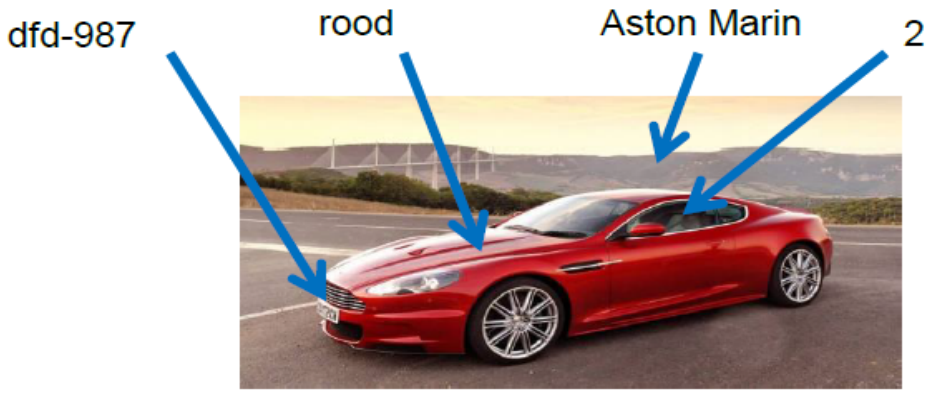
5.2 Bouwstenen relationeel model

5.2.1 Tupel

Een tupel is een geordende lijst met waarden van kenmerken die een object beschrijven. Een tupel is steeds uniek. De waarden in een tupel beschrijven de kenmerken van het object dat voorgesteld wordt door de tupel.

Voorbeeld:

Een tupel voor het beschrijven van een auto: (dfd-987, rood, Aston Martin, 2). Zo



Figuur 5.3: Beschrijving van een rode 2-deurs Aston Martin met nummerplaat dfd-987

zien we dat dit tupel een auto (zie figuur 5.3) voorstelt en dat deze auto vier kenmerken heeft waarvan de waarden 'dfd-987', 'rood', 'Aston Martin' en '2 zijn. Een kenmerk noemen we ook een attribuut.

5.2.2 Attribuut

Een attribuut is een benoemd kenmerk van een tupel. Een attribuut kan maar één waarde hebben, namelijk de attribuutwaarde.

Voorbeeld:

Het attribuut *merk*, een benoemd kenmerk van een auto uit de tupel: (dfd-987, rood, Aston Martin, 2). Deze auto kan maar van 1 merk zijn: het attribuut merk mag maar 1 waarde bevatten.

5.2.3 Domein

Een domein is een verzameling van waarden die voor de attributen in de tupels van een relatie mogen gebruikt worden.

Voorbeeld:

Het domein *kleuren* is een verzameling van alle toegelaten kleuren die een auto kan hebben, dit kan bijvoorbeeld afhankelijk zijn van fabrikant tot fabrikant: rood, groen, blauw, ...

Figuur 5.4: Het domein *kleuren*

5.2.4 Relatie

Een relatie is een verzameling van tupels die gelijksoortige objecten beschrijven.

Voorbeeld:

Een verzameling van tupels die autos beschrijven: { (dfd-987, rood, Aston Martin, 2), (xhd-352, groen, Porche,2), (123-klm, zwart, Maybach,2) }.
Een relatie kan je dan ook zien als een gestructureerde verzameling van gegevens. Het zijn gegevens die allemaal betrekking hebben op gelijkaardige objecten (in het voorbeeld: objecten van auto's) en ze komen in groepjes voor (met name per tupel, dus per object), en binnen elk groepje in eenzelfde volgorde (dus de waarden voor eenzelfde type kenmerk staan steeds op dezelfde plaats in zo'n groepje).

Een relatie in het relationele model mag je zeker niet verwarren met het begrip 'relatie' uit het conceptuele model! In een conceptueel model is een relatie een verband tussen entiteittypes, in een relationeel model is een relatie een tabel (zie figuur 5.5).

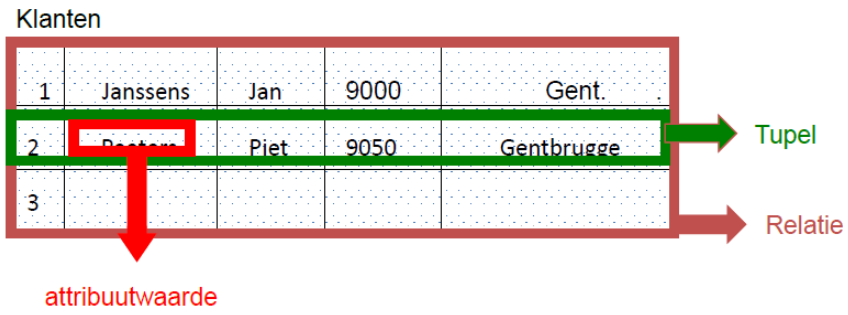
5.2.5 Sleutels

Zoals reeds gezegd moet elk tupel in een relatie uniek zijn. In onderstaande verzameling maakt het attribuut *nummerplaat* elk tupel in deze verzameling uniek.

{(dfd-987, rood, Aston Martin, 2), (xhd-352, groen, rood, 2), (123-klm, zwart, Maybach, 2), (456-aze, geel, Maybach, 2)}

Database		Relationeel	Conceptueel
tabel	→	relatie	entiteittype
rij	→	tupel	entiteit
kolom	→	attribuut	attribuut
kolomwaarde	→	attribuutwaarde	attribuutwaarde

Figuur 5.5: Overzicht benamingen



Figuur 5.6: Voorbeeld benamingen

Hoewel de tupels attributen hebben van dezelfde attribuuttypen, kunnen de attribuutwaarden van twee entiteiten nooit volledig identiek zijn. Attribuuttypen die toelaten om de tupels van elkaar te onderscheiden, worden **sleutelattribuuttype**n genoemd (of kortweg **sleutels**). Men spreekt van kandidaatsleutels, primaire sleutels, alternatieve sleutels, vreemde sleutels.

Kandidaatsleutel

Soms volstaat één attribuuttype om elk tupel in een relatie uniek te identificeren. Zoals in ons voorbeeld heb je aan het attribuuttype nummerplaat genoeg om een auto identiek te identificeren. Soms kan het voorkomen dat je aan één enkel attribuuttype niet genoeg hebt om het tupel te identificeren. Bijvoorbeeld om een kamer in een hotel uniek te identificeren heb je de hotelnaam en het kamernummer nodig. Met enkel het kamernummer kun je uit een reeks hotels nooit uniek een kamer identificeren. Daarom moet de kandidaatsleutel steeds een minimale verzameling van attributen zijn in de tupels van een relatie, waarvan de combinatie elke tupel binnen die relatie uniek kan identificeren.

Het kan voorkomen dat er verschillende attributen of verzamelingen van attributen

een sleutel kunnen vormen. Bijvoorbeeld een student aan de hogeschool kan uniek geïdentificeerd worden aan de hand van het studentennummer en het rijksregisternummer. Beide zijn binnen de school steeds uniek. We hebben dus de mogelijkheid om te kiezen uit 2 sleutels. We noemen deze dan ook de kandidaatsleutels.

Primaire sleutel

Uit één van de kandidaatssleutels wordt de primaire sleutel gekozen. De primaire sleutel moet steeds ingevuld zijn. NULL-waarden zijn dus niet toegelaten. In een relationeel model wordt de primaire sleutel onderstreept.

Alternatieve sleutel

Elke kandidaatsleutel die geen primaire sleutel geworden is, wordt een alternatieve sleutel genoemd.

De alternatieve sleutel moet niet steeds ingevuld zijn (NULL toegelaten) . . . Maar opgelet! Van zodra voor minstens één van de tupels in de relatie deze waarde effectief NULL wordt, verliest de alternatieve sleutel zijn functie van 'sleutel' want tupels met NULL waarden kunnen dan niet meer van elkaar onderscheiden worden via deze alternatieve sleutel.

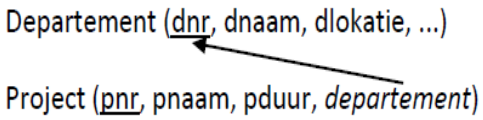
Vreemde sleutel

De vreemde sleutel is een beetje de vreemde eend in de bijt. We gebruiken deze sleutel om verbanden te leggen met andere relaties (tabellen) in ons relationeel model. Een vreemde sleutel heeft niets te maken met de identificatie van de tupel.

Een vreemde sleutel (*foreign key)* is de verbindende schakel tussen twee relaties. Met een waarde uit een rij van de ene relatie kun je in een andere tabel de juiste tupel met gerelateerde gegevens opzoeken.

Voorbeeld:
Een project wordt steeds uitgevoerd in een bepaald departement.

Het relationeel model wordt geschetst in figuur 5.7. Het attribuuttype *departement*



Figuur 5.7: Voorbeeld relationeel model

in de relatie Project is de vreemde sleutel die verwijst naar de sleutel *dnr* in de relatie

Departement. Hierdoor weten we steeds in welk departement een bepaald project uitgevoerd wordt (zie figuur 5.8).
Bij een ER-model zouden we simpelweg een relatie tekenen tussen de twee entiteittypen en kwam er geen vreemde sleutel aan te pas.

Departement	Project
01, productie, plant1	1023, slijpen, *01*
02, onthaal, hoofdgebouw	1024, zagen, *01*
03, administratie, hoofdgebouw	1025, telefonie, *02*
	2089, assembleren, *01*
	0058, lonen_uitbetalen, *03*

Figuur 5.8: Voorbeeld tupels in relationeel model Departement/Project

In een relationeel model noteren we steeds de *integriteitregels* bij een vreemde sleutel:

departement verwijst naar dnr uit Departement en is verplicht

5.3 Vergelijking met het ER-model

- Een relatie is vergelijkbaar met 'entiteittype' en heeft niets te maken met het concept van 'relatie' in het ER-model.
- Een tupel is vergelijkbaar met een 'entiteit'.
- Domein, attribuuttype en attribuutwaarde hebben dezelfde betekenis als in het ER-model.
- Het ER-model biedt ons uitgebreidere mogelijkheden om verbanden voor te stellen. Het relationele model heeft bijna uitsluitend 1-op-veel-verbanden. 1-op-1 verbanden kunnen wel voorgesteld worden namelijk als de primaire sleutel van de ene tabel verwijst naar de primaire sleutel van de andere tabel. Veel-op-veel relaties kunnen echter niet rechtstreeks voorgesteld worden.
- Sleutels hebben ongeveer dezelfde betekenis, op een vreemde sleutel na.

ER Model	Relationeel model
Entiteittype	Relatie (tabel)
Entiteiten	Tupels
Attribuuttype	Attribuuttype
1-1; 1-N; N-N 1ste graad, 2e graad en >	1-1; 1-N,1ste graad en 2de graad (Geen hogere)

Figuur 5.9: Vergelijkende tabel

5.4 Regels van een relationeel model

Uit voorgaande definities kunnen we besluiten dat het relationele model is een logisch model is dat voldoet aan volgende regels:

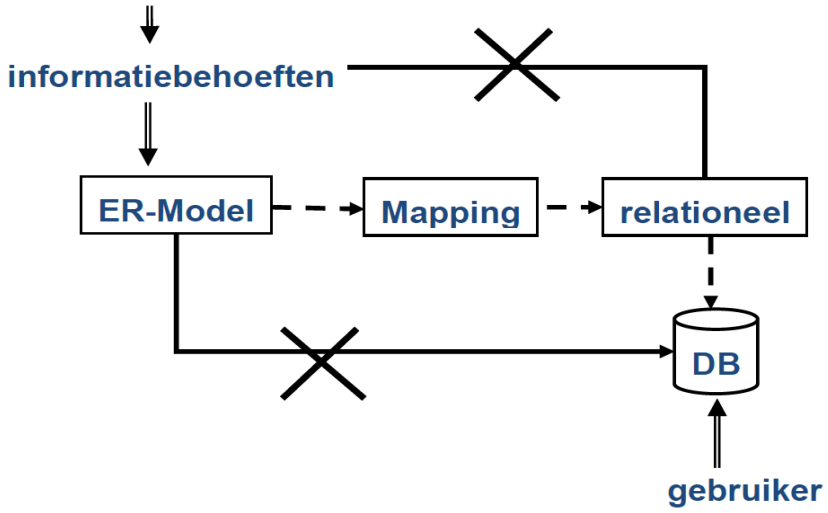
- Elk tupel in een relatie is uniek (elke rij in een tabel is uniek).
- Elk attribuut is enkelvoudig (samengestelde attribuutwaarden zijn niet toegestaan).
- Elk attribuut is enkelwaardig (meerwaardige attribuutwaarden zijn niet toegestaan).

5.5 Mapping

5.5.1 Inleiding

Nu we de nodige termen van het relationeel model besproken hebben kunnen we beginnen met het echte werk, namelijk de mapping van een conceptueel model (ER-model) naar een logisch model (relationeel model). Dit gebeurt in enkele stappen en volgens strikte regels. In principe is mapping, door het volgen van deze stappen en regels, relatief eenvoudig en foutloos uit te voeren.
De stappen zijn de volgende:

1. Elk entiteittype wordt een relatie. Bij het mappen van een EERD waar specialisatie voorkomt, kan het gebeuren dat entiteittypes verdwijnen. Meer uitleg hierover bij



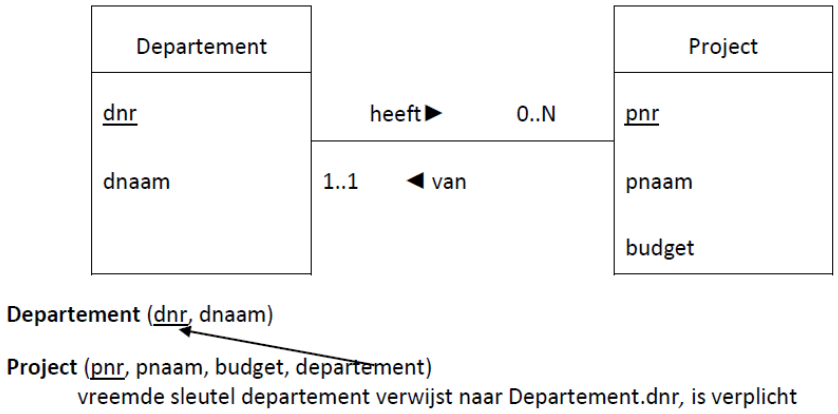
Figuur 5.10: Mapping van een conceptueel model naar logisch model

de mappingregels voor specialisatie.

2. Enkelvoudige attribuuttypes overnemen.
3. Samengestelde attribuuttypes opsplitsen in enkelvoudige attribuuttypes.
4. Primaire sleutel bepalen. Opgelet bij zwakke entiteiten! Zie mapping zwakke entiteiten.
5. Als er nog meerwaardige attributen in het conceptueel model aanwezig zijn, dienen deze in een nieuwe relatie gezet te worden.
6. Voor elke relatie: vreemde sleutel(s) bepalen op volgende wijze:
   - 1-op-N verband: vreemde sleutel in relatie aan N-zijde;
   - 1-op-1 verband: vreemde sleutel in één of andere relatie (afhankelijk van minimumcardinaliteit);
   - veel-op-veel: aparte relatie met beide vreemde sleutels als samengestelde primaire sleutel;
   - unair '1-op-1' of '1-op-veel' verband: vreemde sleutel in zelfde relatie.
7. Bij elke vreemde sleutel de integriteitregels bepalen:
   - naar welke primaire sleutel deze vreemde sleutel verwijst;
   - of de vreemde sleutel verplicht of optioneel is: dit bepaal je aan de hand van de minimumcardinaliteit;
   - eventueel of de vreemde sleutel uniek is (zie verder bij 1-op-1-verbanden).

5.5.2 Mapping van één-op-veel relaties

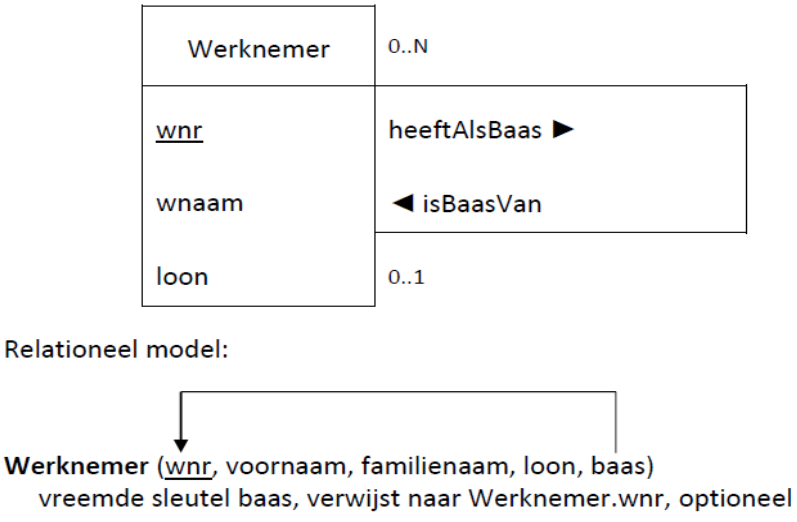
De primaire sleutelattributen van het 'ouder-entiteittype' (entiteittype aan de 1-kant van de relatie) worden als vreemde sleutelattributen toegevoegd aan het 'kind-entiteittype' (entiteittype aan de veel-kant van de relatie).



Figuur 5.11: Voorbeeld mapping

5.5.3 Mapping van één-op-veel recursieve (unaire) relaties

Volg dezelfde regels als voor een normale 1-op-veel relatie. Behalve dat nu de vreemde sleutel in dezelfde relatie zal staan. Als naam voor de vreemde sleutel kies je de rolnaam aan de 1-kant van de relatie (zie figuur 5.12).



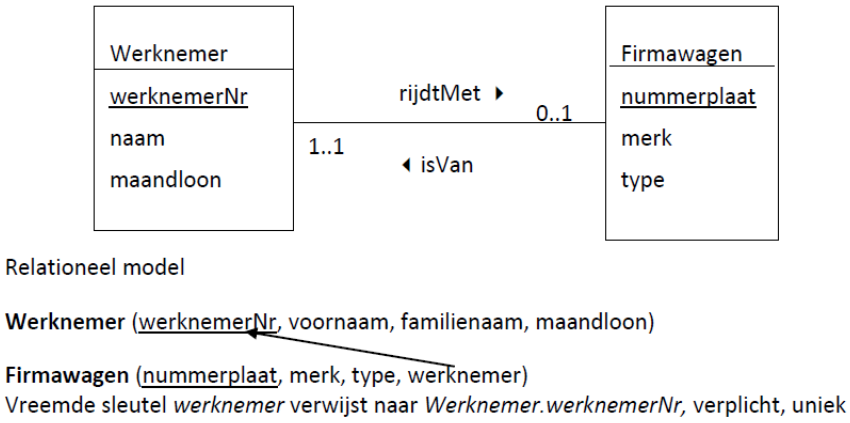
Figuur 5.12: Voorbeeld mapping in recusieve relatie

5.5.4 Mapping van één-op-één relaties

We baseren ons op de participatieconstraint om te beslissen of we beiden samenvoegen tot één relatie of beide relaties behouden waarbij we de primaire sleutel kopiëren. Op basis van de volgende participatieconstraints maken we onze beslissing:

1. Verplichte deelname aan één zijde van de 1-op-1 relatie.

   Het entiteittype met de optionele participatie is de ouder, die met de verplichte participatie het kind. Er wordt dan een verband gelegd tussen beide relaties door de primaire sleutel van de ouder (verplichte deelname) te kopiëren naar het kind (optionele deelname). (zie 5.13)



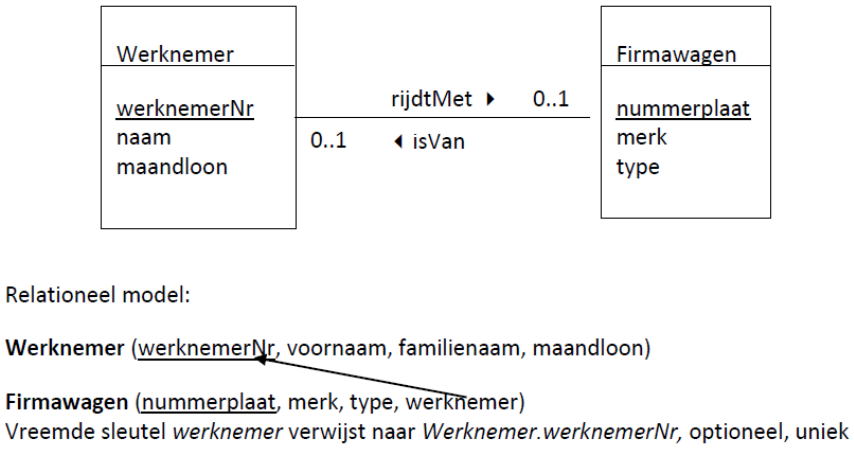
Figuur 5.13: Voorbeeld mapping bij verplichte deelname aan één zijde van de 1-op-1 relatie

2. Optionele deelname aan beide zijdes van de één-op-één relatie.

   In dit geval kan je kiezen in welke relatie de vreemde sleutel zal opgenomen worden, tenzij je weet welke entiteiten de grootste kans maken op verplichte participatie.

   Indien je aanneemt dat het grootste deel van de auto's, maar niet allemaal, gebruikt wordt door werknemers maar dat slechts een klein deel van de werknemers met een auto rijdt, kan je besluiten dat het Firmawagen entiteittype dichter aanleunt bij mandatory dan Werknemer en dat de primaire sleutel van Werknemer zal gekopieerd worden naar Firmawagen.(zie 5.14)

   Indien je bij werknemer een vreemde sleutel zou opnemen, die verwijst naar de wagen dan heb je veel meer kans op NULL-waarden in de vreemde sleutel. Bovenstaande oplossing is bijgevolg beter omdat je weet dat bijna elke auto aan



Figuur 5.14: Voorbeeld mapping bij optionele deelname aan beide zijdes van de 1-op-1 relatie

een werknemer toebehoort en je bijgevolg weinig NULL-waarden zal hebben in de vreemde sleutel.

5.5.5 Mapping van één-op-één unaire (recursieve) relaties

Volg dezelfde regels als voor een normale n-op-n relatie, werk gelijkaardig zoals bij de één-op-veel recursieve relaties.

5.5.6 Mapping van veel-op-veel (N:N) relaties

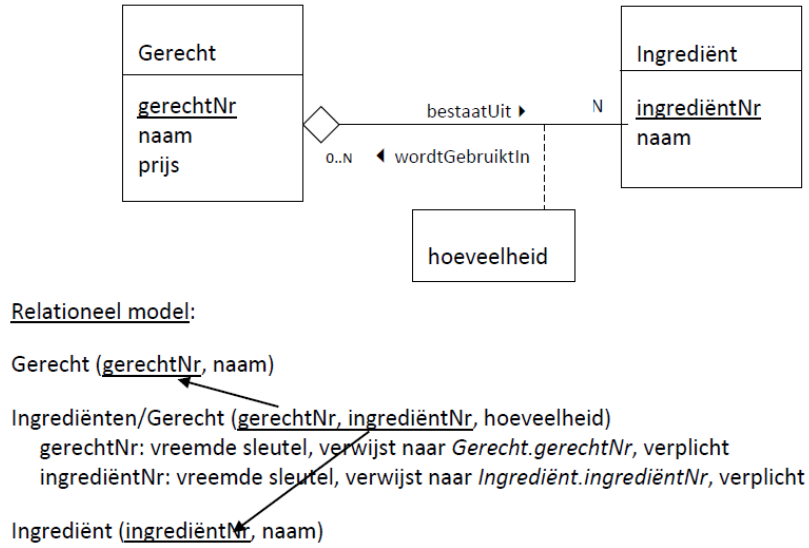
Maak voor elke veel-op-veel relatie uit het conceptuele model een nieuwe relatie aan in het relationeel model en kopieer voor elke deelnemende entiteit de primaire sleutelattributen als vreemde sleutels. De combinatie van beide vreemde sleutels vormt de primaire sleutel in deze nieuwe relatie. Neem ook de relatie-attributen over in de nieuwe relatie (zie figuur 5.15).

5.5.7 Mapping van zwakke entiteittypen

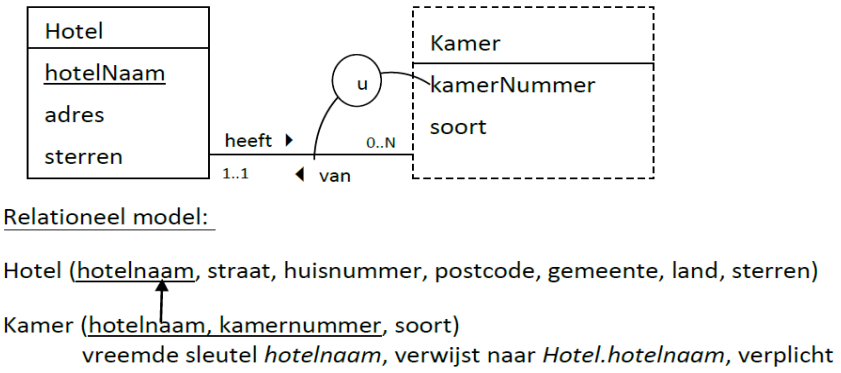
Elk entiteittype, ook het zwakke, wordt een nieuwe relatie. De sleutel van de relatie, die het zwak entiteittype voorstelt, is een combinatie van de sleutel van het entiteittype waarvan het bestaansafhankelijk is en eventueel een partiële sleutel van het zwakke entiteittype. Zie voorbeelden in figuur 5.16 en figuur 5.17.

5.5.8 Mapping van meerwaardige attributen

Een meerwaardig attribuut uit het conceptueel model zal resulteren in een nieuwe relatie in het relationeel model met een betekenisloze sleutel.
Opmerking: een betekenisloze sleutel is een sleutel die eigenlijk niets zegt over het



Figuur 5.15: Voorbeeld mapping bij veel-op-veel relatie

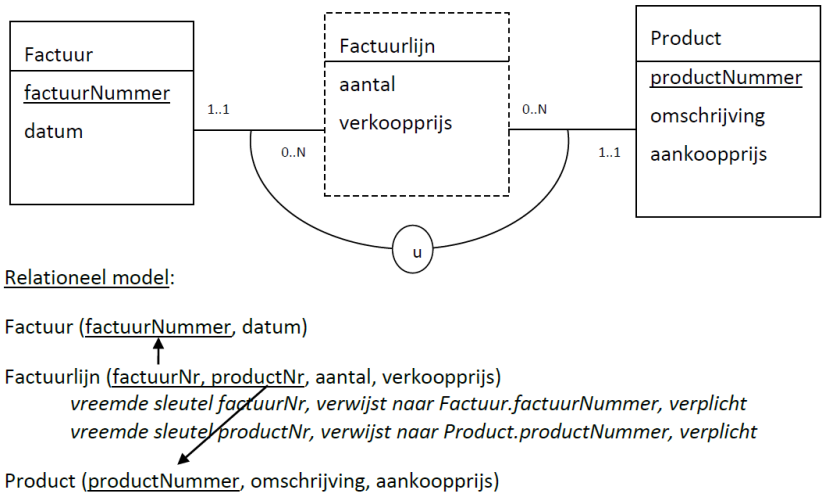


Figuur 5.16: Voorbeeld 1: mapping bij zwakke entiteiten

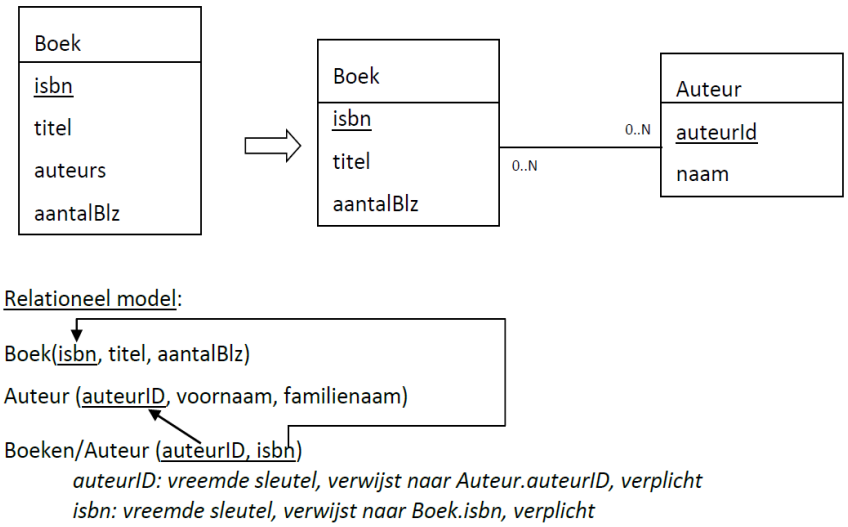
entiteittype, deze dient enkel om een entiteit uniek te maken. Dit zal meestal een automatisch oplopend nummer zijn.

5.5.9 Keuze van de primaire sleutel

In het Entity Relationship Model is voor sterke entiteittypen de sleutel aangeduid. Meestal hoeven we in het relationeel model die sleutel gewoon over te nemen. Soms kiezen we echter om een andere kandidaatsleutel te gebruiken of om een nieuwe sleutel te creëren. We doen dit omdat in het relationeel model de primaire sleutel gebruikt wordt als referentie in andere tabellen (relaties). Indien de sleutel dan bestaat uit een grote hoeveelheid tekst, of indien die sleutel bestaat uit een groot aantal attributen, moet



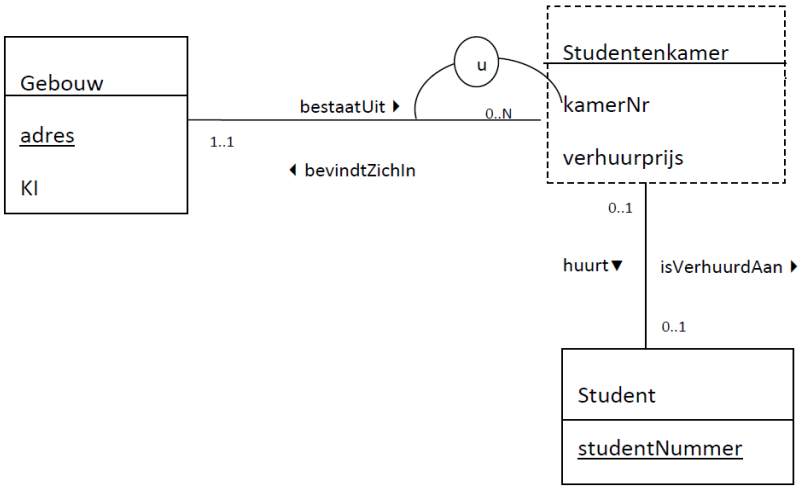
Figuur 5.17: Voorbeeld 2: mapping bij zwakke entiteiten



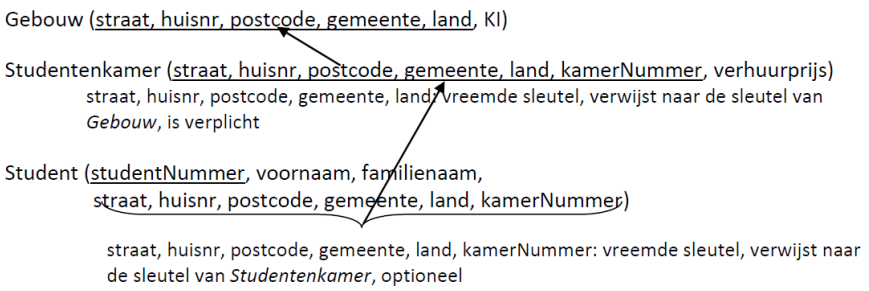
Figuur 5.18: Voorbeeld mapping van meerwaardige attributen

die sleutel volledig gebruikt worden als vreemde sleutel. Dit kan nadelig zijn voor de performantie bij het opzoeken van de gegevens in de relationele databank.

Neem als voorbeeld het conceptueel model in figuur 5.19. Indien we hier in het relationeel model *adres* zouden overnemen als primaire sleutel in de relatie Gebouw dan zouden we het relationeel model zoals geschreven in figuur 5.20 verkrijgen (indien we aannemen dat de meeste studenten in de databank een kamer huren).



Figuur 5.19: Voorbeeld i.v.m. keuze primaire sleutel

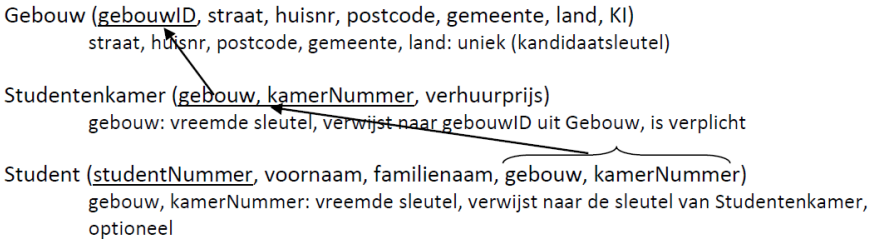


Figuur 5.20: Voorbeeld i.v.m. keuze primaire sleutel: oorspronkelijk relationeel model

Telkens we naar een gebouw willen refereren moeten we de volledige sleutel van Gebouw gebruiken: straat + huisnr + postcode + gemeente + land. In zo'n geval is het aan te raden om een andere primaire sleutel te kiezen voor Gebouw. Aangezien er verder geen kandidaatsleutels aanwezig zijn (KI is niet uniek) kiezen we voor een nieuwe betekenisloze sleutel (gebouwID). Het nieuwe relationele model zien we in figuur 5.21.

Het is aan te raden om in het relationele model een integriteitregel toe te voegen zodanig dat we achteraf bij het maken van het fysiek model nog weten dat het adres van een gebouw uniek is (zie volgend hoofdstuk fysiek model).

Opmerking: waarom niet meteen in het conceptueel model een 'nummer' of 'ID' voorzien?



Figuur 5.21: Voorbeeld i.v.m. keuze primaire sleutel: nieuw relationeel model

Dit zou conceptueel gezien totaal fout zijn! Het conceptueel model is een weerspiegeling van de werkelijkheid. In de werkelijkheid wordt een gebouw geïdentificeerd aan de hand van het adres. Indien we het conceptueel model zouden vertalen naar een hiërarchisch model of een OO-model hebben we dat nummer nergens voor nodig. Het conceptueel model moet conceptueel blijven, moet kunnen gemapt worden naar gelijk welk logisch model (een hiërarchisch model of een OO-model of een relationeel model).

5.6 Mapping van specialisatie

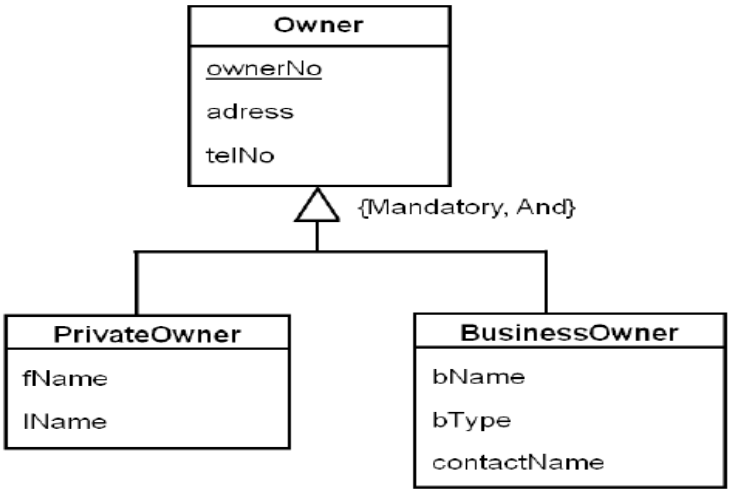
Er zijn verschillende mogelijkheden bij het omzetten van supertype/subtype verbanden. De mogelijkheden hangen af van de disjoint- en participatie-constraints. Voor elke mogelijke combinatie van de soorten disjoint- en participatie-contraints is er een specifieke mappingregel.
Spijtig genoeg is het perfect mappen van specialisatie niet mogelijk. Elke mogelijke regel zorgt ervoor dat de oorspronkelijke situatie zo goed mogelijk benaderd wordt. Toch zal je merken dat we soms enkele restricties verliezen of via instellingen in de databank het probleem moeten oplossen. Noteer dit in een integriteitregel!

5.6.1 Mapping van Mandatory, And

Neem als voorbeeld het conceptueel model uit figuur 5.22. Bij de mapping van (mandatory, and) stellen we één relatie op die een combinatie is van het supertype en alle subtypes. Om aan te duiden over welk subtype het gaat gebruiken we de attributen *isPrivateOwner* en *isBusinessOwner*. Deze zijn van het type boolean. Wanneer bijvoorbeeld het attribuut isPrivateOwner de waarde "true" krijgt, kan men zeggen dat de eigenaar een private owner is. Deze vlaggen geven dus aan van welk type een owner is. Vandaar dat we dit ook wel *"flags"* of vlaggen noemen.

Op die manier komen we tot volgend relationeel model:

Owner(ownerNo, street, hnr, postalCode, city, telNo, fName, lName, bName, bType, contactname, **isPrivateOwner**, **IsBusinessOwner**)



Figuur 5.22: Voorbeeld Mandatory, And

Door aan beide vlaggen de waarde "true" toe te kennen, kunnen we een private owner met een business owner combineren. De restrictie "AND" blijft gehandhaafd.

Een probleem stelt zich bij de mapping van de restrictie "mandatory". Het is mogelijk aan beide vlaggen de waarde "false" toe te kennen. In dit geval is de eigenaar (owner) noch private, noch business waardoor niet langer voldaan is aan de restrictie "mandatory". Dit is op te lossen door in de databank in te stellen dat minstens één van beide velden de waarde "true" moet hebben. Op deze manier blijft de restrictie "mandatory" ook gehandhaafd. Stel in het relationeel model hiervoor een integriteitsregel op: *isPrivateOwner of isBusinessOwner moet 'true' zijn*.

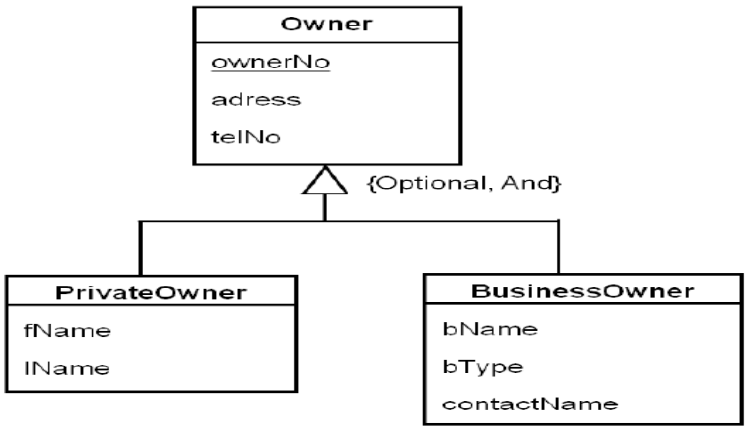
5.6.2 Mapping van Optional, And

Neem als voorbeeld het conceptueel model uit figuur 5.23.

Voor de mapping van (optional, and) stellen we steeds twee relaties op. Eén voor het supertype en één die een combinatie is van alle subtypes. De restrictie "AND" implementeren we op net dezelfde manier als in de vorige mapping. Ook hier gebruiken we weer vlaggen (booleans) die aanduiden om welk subtype het gaat.

Het relationeel model ziet er dus als volgt uit:

Owner(ownerNo, street, hnr, postalCode, city, telNo)
OwnerDetails(ownerNo, fName, lName, bName, bType, contactname, **isPrivateOwner**, **IsBusinessOwner**)



Figuur 5.23: Voorbeeld Optional, And

ownerNo: vreemde sleutel die verwijst naar Owner.ownerNo, verplicht.
isPrivateOwner: boolean
isBusinessOwner: boolean

Het verschil met de vorige mapping is dat er nu geen restrictie "Mandatory", maar "Optional" is. Dit lossen we op met de supertype-relatie (owner). Elke eigenaar komt in deze relatie terecht. Wanneer hij/zij/het noch een private owner, noch een business owner is, blijft het hierbij. In al de andere gevallen komt er nog een tupel bij in de relatie OwnerDetails met de nodige vlaggen.

Via deze mapping kunnen we nu beide restricties handhaven in het relationeel model.

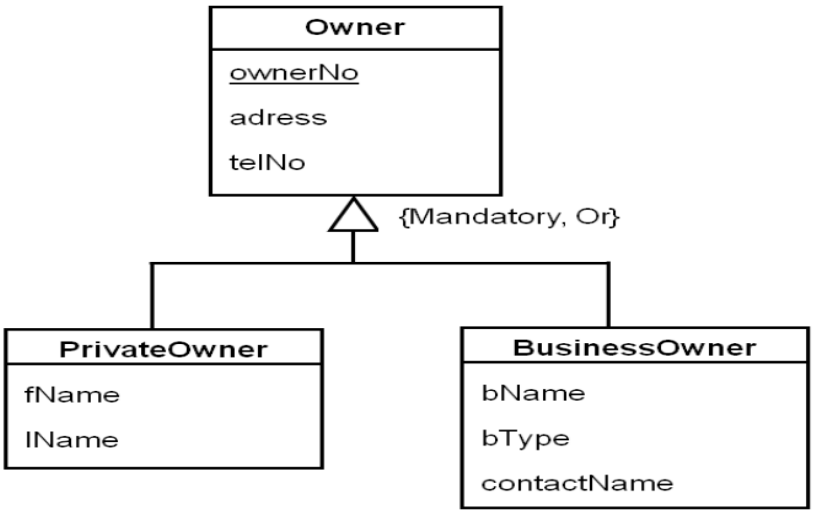
5.6.3 Mapping van Mandatory, Or

Neem als voorbeeld het conceptueel model uit figuur 5.24.

In deze situatie wordt voor elk subtype een aparte relatie aangemaakt. Een relatie voor het supertype is er niet. Daarom moet elke relatie alle attributen van het supertype bevatten, bovenop de subtype specifieke attributen.

Het relationeel model ziet er dus als volgt uit:

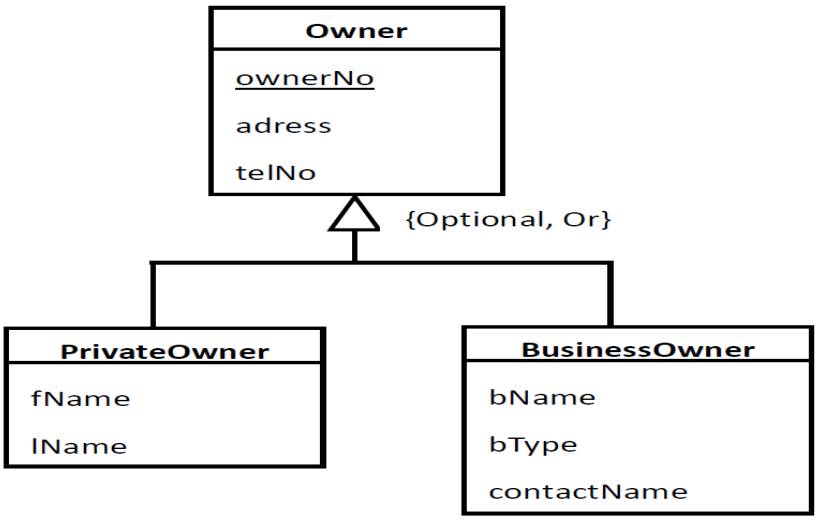
PrivateOwner(pOwnerNo, fName, lName, street, hnr, postalCode, city, telNo)
BusninessOwner(bOwnerNo, bName, bType, contactName, address, telNo)



Figuur 5.24: Voorbeeld Mandatory, Or

Op deze manier is het onmogelijk een eigenaar aan de databank toe te voegen, zonder dat deze tot één en maximum één van de subtypes behoort. Alle restricties blijven behouden.

5.6.4 Mapping van Optional, Or



Figuur 5.25: Voorbeeld Optional, Or

Neem als voorbeeld het conceptueel model uit figuur 5.25.

Bij deze mapping wordt een aparte relatie voor het supertype en elk subtype opgesteld. De relaties die een subtype voorstellen bevatten enkel de attributen die specifiek bij dat bepaald subtype horen. Op deze manier is het mogelijk om een eigenaar toe te voegen die noch private, noch business is. De restrictie "Optional" blijft dus behouden. De restrictie "OR" daartegen verliezen we. Het is mogelijk om een eigenaar toe te voegen aan zowel het supertype en beide subtypes. Om die reden kan de restrictie "OR" niet gegarandeerd worden.

Het relationeel model ziet er dus als volgt uit:

Owner(ownerNo, street, hnr, postalCode, city, telNo)

PrivateOwner(ownerNo, fName, lName)
ownerNo: vreemde sleutel die verwijst naar Owner.ownerNo, verplicht.

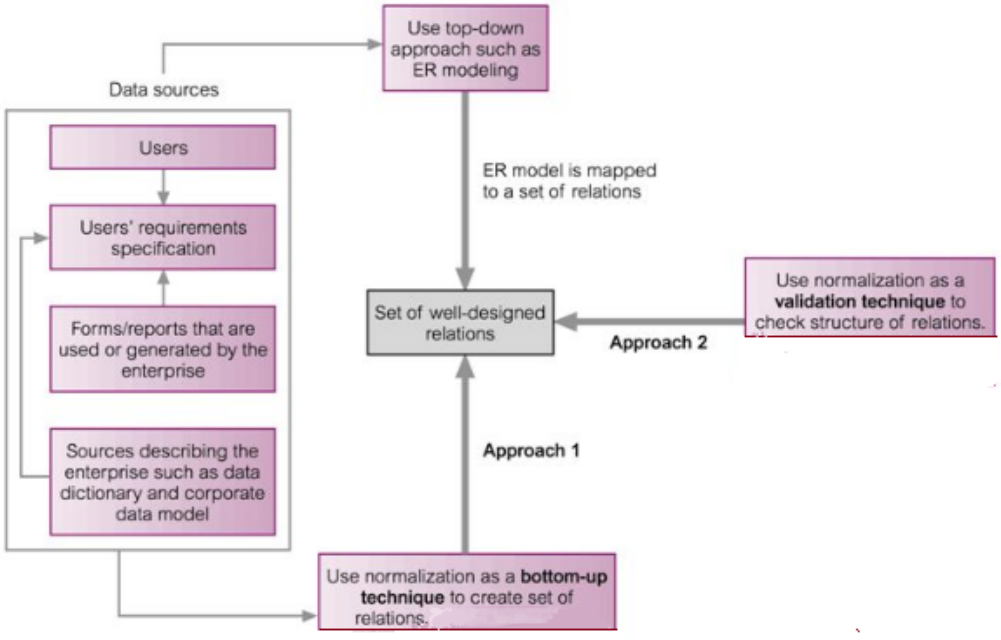
BusninessOwner(ownerNo, bName, bType, contactName)
ownerNo: vreemde sleutel die verwijst naar Owner.ownerNo, verplicht.

HOOFDSTUK 6

Normalisatie

6.1 Inleiding

Het opstellen van een datamodel kan top down gebeuren. In dat geval vertrekt men van objecttypen en hun verbanden (associaties) en voegt men daarna de attribuuttypes toe.

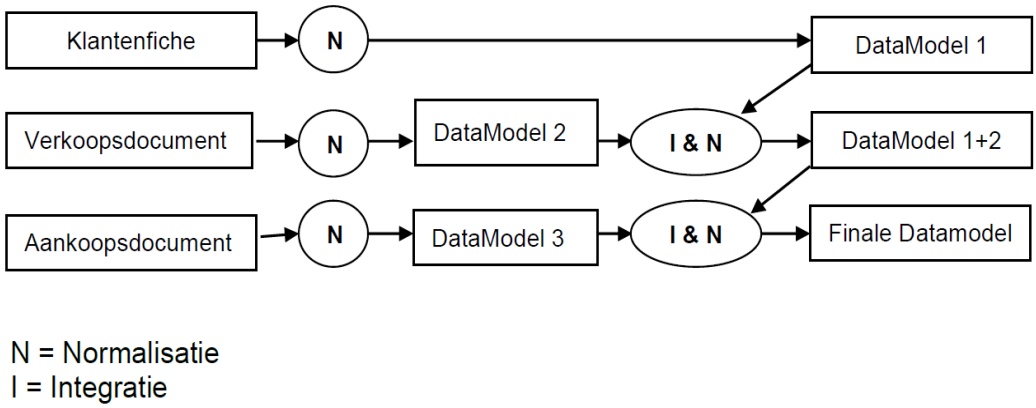


Figuur 6.1: Twee mogelijke benaderingen om tot een correct relationeel model te komen.

Het opstellen van het Entity Relationship Model werkt op deze manier. Na mapping bekomt men een set van relaties (tabellen).

Maar er is nog een andere benadering, namelijk de bottom up techniek. Hierbij vertrekt men van de attributen die gegroepeerd worden tot objecttypen. De in dit hoofdstuk beschreven normalisatiemethode werkt zo. De normalisatiemethode is de oudste manier om een datamodel op te stellen. Deze methode werd ontwikkeld door Codd, die ook de grondlegger was van de relationele databank.

Normalisatie vertrekt van gegevensverzamelingen (documenten, schermen,) die tot



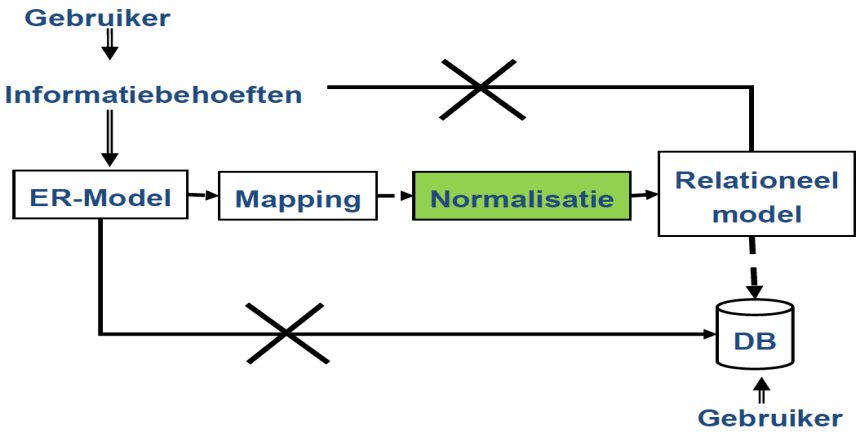
Figuur 6.2: Normalisatie en integratie vanuit documenten.

een relationeel datamodel genormaliseerd worden. Deze verschillende datamodellen, bekomen uit de verschillende documenten, worden dan tot één datamodel geïntegreerd. (zie schema 6.2)

Een datamodel kan dus volledig door middel van normalisatie opgebouwd worden.

Men kan evenwel normalisatie ook gebruiken om een datamodel, opgesteld zoals we deden in vorige hoofdstukken, te controleren.

Het opstellen van een datamodel door middel van normalisatie wordt de eigenlijk niet meer zo vaak gedaan. Men prefereert het opstellen van een conceptueel model dat daarna gemapt wordt naar een relationeel model. Dat model wordt gecontroleerd door middel van de normalisatietechniek en zo bekomt men uiteindelijk een genormaliseerd relationeel model. Om deze controle correct te kunnen uitvoeren is het essentieel dat je elke stap van de normalisatietechniek snapt en beheerst. Dit is slechts mogelijk nadat je een datamodel kan opstellen op basis van documenten door middel van normalisatie.



Figuur 6.3: Normalisatie als stap na Mapping

6.1.1 Doel

Het doel van normalisatie is het bekomen van een set van relaties (tabellen) die voldoen aan de regels van het relationele model en waar geen *redundantie* meer in voorkomt.

Redundantie betekent dat bepaalde informatie op meerdere plaatsen opgeslagen wordt. De voornaamste nadelen waarom we redundantie willen vermijden, zijn:

- risico's van inconsistentie: indien dezelfde informatie op meer dan één plaats tegelijk aanwezig is, dan kunnen de verschillende manieren van informatieopslag elkaar tegenspreken;
- anomalieën bij het gebruik van de data (hierover later meer).

Beschouw de relatie in figuur 6.4. In deze relatie vinden we bijvoorbeeld het feit dat de afdeling (branche) B003 als adres "22 Deer Rd, London" heeft, drie maal terug. Het is dus duidelijk dat in deze tabel dezelfde feiten meermaals gestockeerd worden. Men zegt dat deze relatie **redundantie** vertoont. Zoals eerder gezegd kan deze redundantie enkele problemen veroorzaken, waaronder anomalieën.

6.1.2 Anomaliëen

Enkele voorbeelden:

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Figuur 6.4: Tabel PersoneelAfdeling

- Wat als je in het voorgaand voorbeeld een afdeling (=branche) wil toevoegen zonder stafleden?
  Dit gaat niet!
- Wat als je rij SA9, Mary Howe, Assistant, 9000, B007, 16 Argyll St, Aberdeen verwijdert?
  Dan ben je alle informatie over branchNo B007 kwijt aangezien Mary Howe het enig staflid is dat in die branche werkt.

Deze zaken noemen we anomalieën. Er zijn verschillende soorten anomalieën die we moeten kunnen herkennen en voorkomen. We bespreken de invoeg-, verwijder- en modificatieanomalie.

Invoeganomalie
Is het toevoegen van een branche zonder stafleden mogelijk?
Omdat men enkel gegevens over een branche samen met een staflid kan bewaren, is het onmogelijk om in de huidige "relatie" een branche zonder stafleden toe te voegen. Terwijl dit in het echte leven misschien wel de normale gang van zaken is.

Wat als je een staflid aan een bestaande branche toevoegt, maar een ander adres opgeeft? Omdat bij elk staflid de informatie van een branche herhaald wordt, is het mogelijk om inconsistente data in de databank te genereren. Wanneer men bij een nieuw staflid een bestaand branchenummer gebruikt maar een ander adres opgeeft, creëren we inconsistente data.

Verwijderanomalie
Wat als we het laatste lid van een branche verwijderen?

Dan verlies je alle informatie omtrent die branche. Als je een tupel verwijdert en daarmee ook alle informatie over een ander object dan spreekt men van een verwijderanomalie.

Modificatieanomalie
Wat als we bij een staflid het adres van de branche aanpassen, maar bij de andere stafleden niet?
Ook dit zorgt voor inconsistente data en spreken we van een modificatieanomalie.

Oplossing Om deze anomalieën te voorkomen is het nodig de gegevensverzameling "StaffBranch" te splitsen in twee relaties namelijk **Branch** en **Staff**. (zie figuur 6.5)
Hierdoor kunnen we met de data werken (invoegen, verwijderen, wijzigen) zonder dat

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Figuur 6.5: Twee aparte (gescheiden) relaties Branch en Staff

er een anomalie optreedt.

Deze opsplitsing noemt men normalisatie van de gegevensverzameling. Het grote voordeel van een genormaliseerd gegevensmodel is dat in een databank die gestructureerd is volgens zo'n model elk gegeven slechts één keer voorkomt. Er is dus geen redundantie en bijgevolg minder kans op inconsistenties en anomalieën. Bij wijziging van een gegeven (vb. een branche verhuist en verandert dus van adres) hoef je in zo'n databank dan ook maar op één plaats die wijziging door te voeren (m.a.w. het adres van een branche komt maar één keer voor in de databank).

Je kan normalisatie niet zomaar op goed gevoel gaan uitvoeren. Het is nodig dat je volgens enkele regels een bepaald proces volgt. Deze regels steunen op het principe van ***functionele afhankelijkheid***.

6.1.3 Functionele afhankelijkheid

Definitie: B is functioneel afhankelijk van A (kan samengesteld zijn) als er met een

waarde van A ten hoogste één waarde van B overeenkomt.

Of anders geformuleerd: als de waarde van A gekend is, dan is ook de waarde van B gekend. In feite komt het erop neer dat wanneer je de waarde van het attribuuttype A (of combinatie van attribuuttypes) in een tupel kent, je de waarde van de functioneel afhankelijke attribuuttypes in die tupel éénduidig kunt bepalen of terugvinden.
Notatiewijze: A $\rightarrow$ B

We lezen: A bepaalt B

of B is functioneel afhankelijk van A.

A noemen we de ***determinant***.

Beschouw in de tabel Staff in figuur 6.6 de attribuuttypes 'staffNo' en 'position'. Neem

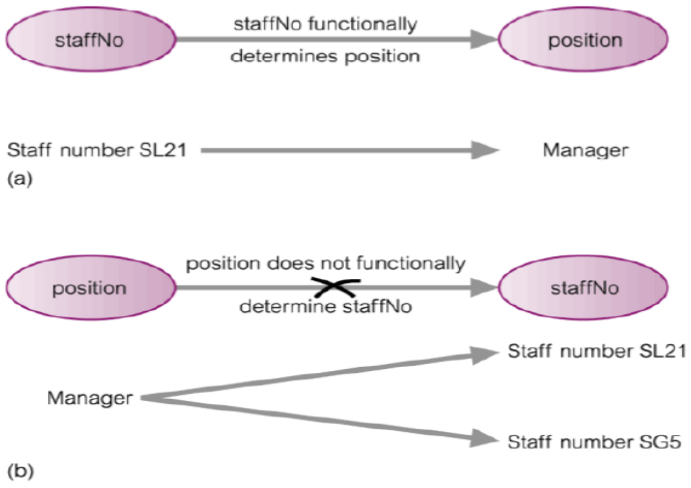
Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Figuur 6.6: Tabel Staff

aan dat iedere stafmedewerker een uniek stafnummer (staffNo) heeft gekregen. Dan geldt dat staffNo $\rightarrow$ position, maar niet dat position $\rightarrow$ staffNo.
Dus bij één waarde van staffNo hoort ook één waarde van position, maar het omgekeerde is niet waar. Bijvoorbeeld, als gesproken wordt over de stafmedewerker met nummer SL21, dan gaat het over John White, een manager. Als gesproken wordt over de stafmedewerker "de manager", dan is het onmogelijk te bepalen welke stafmedewerker bedoeld wordt. Er zijn namelijk twee managers. Dus door het opgeven van een position is het niet mogelijk het staffNo uniek te bepalen, bijgevolg kan er ook geen specifieke persoon aan gekoppeld worden.

Partiële afhankelijkheid



Figuur 6.7: Functionele afhankelijkheid

Definitie: B is partieel afhankelijk van A als A samengesteld is en als er met één waarde van A ten hoogste één waarde van B overeenkomt, maar niet elke deelsleutel van A is nodig om B te bepalen.

A bevat dus een **overbodig attribuut** om B te bepalen.

Voorbeeld:

Stel je hebt de relatie zoals te zien in tabel 6.1.

Nr	Naam	Beroep
10	Peter	dokter
11	**Peter**	**docent**
12	Jan	kunstenaar
13	Bart	docent

Tabel 6.1: Partiële afhankelijkheid

Er kan gezegd worden dat (nr, naam) $\rightarrow$ beroep. Met (nr, naam) kan er steeds één bepaald beroep aangeduid worden. Alleen is het attribuuttype 'naam' in de determinant overbodig. Beroep is namelijk enkel maar afhankelijk van het attribuuttype 'nr'. Met enkel het attribuut 'nr' kan nog steeds uniek een beroep bepaald worden. Het extra attribuuttype 'naam' is hiervoor niet nodig. Of anders gezegd: beroep is partieel afhankelijk van (nr, naam).

Volledige functionele afhankelijkheid

Definitie: B is volledig afhankelijk van A als geldt:
Met 1 waarde van A, wordt hoogstens 1 waarde van B gevonden en er zijn geen partiële en geen transitieve afhankelijkheden meer.

Transitieve functionele afhankelijkheid

Definitie: C is transitief afhankelijk van A als geldt:
(A $\rightarrow$ B) en (B $\rightarrow$ C) en niet (B $\rightarrow$ A)

Wanneer er een A $\rightarrow$ B bestaat en een B $\rightarrow$ C, dan kunnen we met A via B uniek de waarde van C bepalen. Dit op voorwaarde dat B $\rightarrow$ A niet geldt.

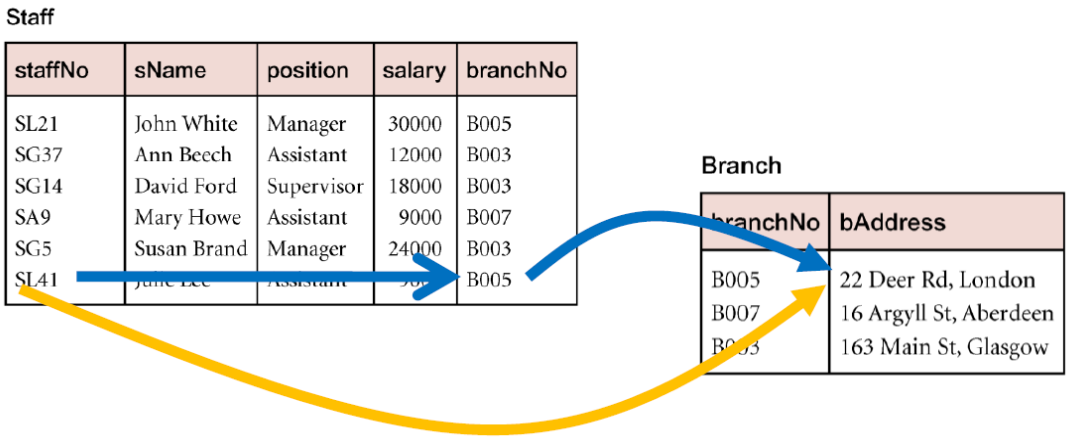
Voorbeeld:

In figuur 6.8 geldt:

staffNo $\rightarrow$ branchNo, en branchNo $\rightarrow$ bAdress en niet (branchNo $\rightarrow$ staffNo).

Als je het staffNo van een stafmedewerker weet, kan je ook uniek bepalen op welk adres hij werkt. Ook al is het adres niet rechtstreeks afhankelijk van het staffNo.

We zeggen dat bAdress transitief afhankelijk is van staffNo.



Figuur 6.8: Transitieve afhankelijkheid

Identificeren van Functionele Afhankelijkheden

Het identificeren van de functionele afhankelijkheden tussen de verschillende attribuuttypes kan enkel als je het probleemgebied volledig begrijpt. M.a.w. aan de hand van informatie van de gebruiker(s) van de databank, reeds bestaande documenten en formulieren moet je proberen te achterhalen hoe de informatie bepaald wordt.

Op basis van het formulier in 6.9kunnen volgende afhankelijkheden afgeleid wor-

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

Figuur 6.9: Identificeren van functionele afhankelijkheden

den:

staffNo $\rightarrow$ sName, position, salary, branchNo, bAddress

sName $\rightarrow$ staffNo, position, salary, branchNo, bAddress

branchNo $\rightarrow$ bAddress

bAddress $\rightarrow$ branchNo

Dit heb je bekomen door voor elke kolom in het document je af te vragen hoe dit gegeven bepaald wordt.

Na het praten met de opdrachtgever komen we te weten dat het zeer waarschijnlijk is dat er nog extra mensen aangenomen zullen worden. Het is mogelijk dat hierbij mensen zitten met dezelfde naam als een andere werknemer. Hoewel met de huidige data onze functionele afhankelijkheden correct zijn, moeten we rekening houden met deze informatie. Dit zorgt ervoor dat de functionele afhankelijkheid ”sName $\rightarrow$ staffNo, position, salary, branchNo, bAddress” moet verdwijnen.

Ook vragen we ons af of op één bAddress werkelijk maar één branch kan gehuisvest zijn. Dit blijkt ook niet het geval te zijn. Er bestaat namelijk een branch B002, waarvoor nog geen personeel is aangenomen en die bijgevolg niet op dit document afgeprint is.

Dit is iets waar je zeker voor moet opletten! Documenten geven meestal niet alle informatie weer. Slechts door het bestuderen van het 'probleemgebied' (waar zich de mensen bevinden die de databank zullen gebruiken) kan de juiste informatie achterhaald worden.

Hierbij blijven dan volgende afhankelijkheden over:

staffNo $\rightarrow$ sName, position, salary, branchNo, bAddress branchNo $\rightarrow$ bAddress

Deze afhankelijkheden zullen als basis gebruikt worden voor het bepalen van de primaire sleutel. Alle gevonden determinanten zijn namelijk kandidaatsleutels.

6.2 Normalisatieproces

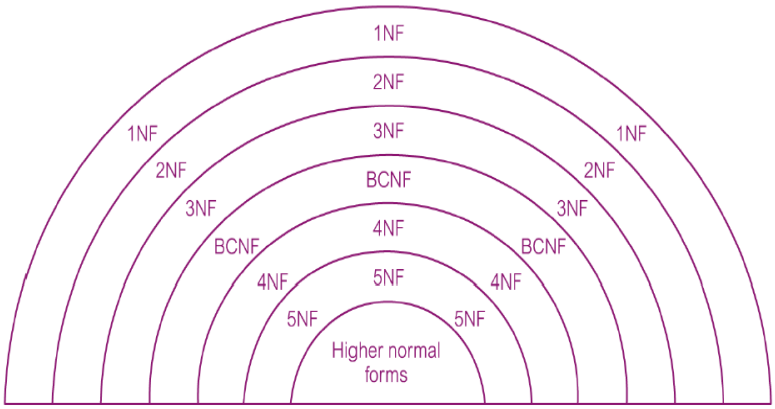
Je begint met het opsommen van alle gegevens die in de databank moeten komen. Al die gegevens (attribuuttypes) stop je in één verzameling. Door het toepassen van normalisatie ga je dan deze attribuuttypes herverdelen over meerdere relaties. Je herorganiseert met andere woorden de initieel ongestructureerde gegevensverzameling totdat je een gegevensverzameling in relationele vorm bekomt. Zoals eerder vermeld, is dit een stapsgewijs proces. Het eindresultaat is een genormaliseerd relationeel gegevensmodel, d.w.z. een geheel van aan elkaar gekoppelde relaties, waarbij de attribuuttypes van elke relatie een logisch geheel vormen.

In figuur 6.10 kan je de Normalisatiestappen van Codd terugvinden. Deze worden uitgevoerd van buiten naar binnen. Zoals je ziet kan je redelijk ver gaan, in deze cursus beperken we ons tot de eerste drie normalisatiestappen.

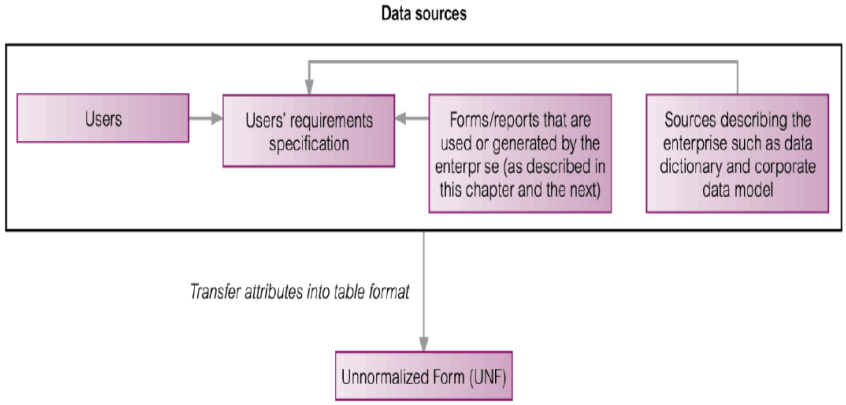
Voorbereiding: verzamel de nodige gegevens doormiddel van gesprekken en documentatie.

We vertrekken van een ongenormaliseerde gegevensverzameling. We zeggen dat deze zich in de nulde normaalvorm bevindt (0NV). Dit is één grote verzameling (relatie) van alle gegevens (attribuuttypes) uit de onderzochte informatiebron (document, scherm, formulier, ...), die we verder opsplitsen (decompositie) tot verschillende delen (relaties).

Voorbeeld: We vertrekken van de documenten zoals getoond in figuur 6.12. Het betreft hier een overzicht van de bestellingen per klant in een meubelbedrijf.



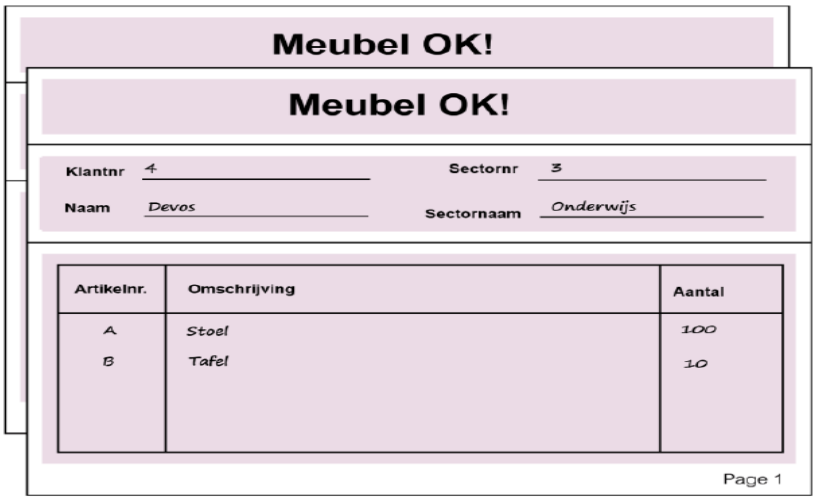
Figuur 6.10: Normalisatiestappen van Codd



Figuur 6.11: Voorbereiding

Uit een gesprek met de eigenaar leren we volgende zaken:

- een klant behoort steeds tot één sector;
- per bestelling kunnen meerdere artikelen besteld worden;
- klanten hebben elk een uniek klantnummer;
- sectoren worden geïdentificeerd aan de hand van een sectornummer;
- artikelen worden geïdentificeerd aan de hand van een artikelnummer.



Figuur 6.12: Overzicht bestellingen per klant

Stel R0 op en bepaal de functionele afhankelijkheden

Op basis van de documenten en deze informatie kunnen we de ongenormaliseerde

nr	klantNaam	sectorNr	sectorOmschr	artikelNr	artikelOmschr	aantal
1	Janssens	1	Medische sector	A B C	Stoel Tafel Kast	100 25 8
2	Peeters	2	Automechanische sector	A D	Stoel Boekenkast	150 25
3	Lemmens	2	Automechanische sector	A	Stoel	50
4	Devos	3	Onderwijs	A B	Stoel Tafel	100 10
5	Mertenss	1	Medische sector	B	Tafel	12

Tabel 6.2: Bestelling

tabel (6.2) opstellen en volgende functionele afhankelijkheden afleiden:

nr $\rightarrow$ klantnaam, sectorNr, sectorOmschrijving

sectorNr $\rightarrow$ sectorOmschr

nr, artikelNr $\rightarrow$ artikelOmschr, aantal

artikelNr → artikelOmschr

Schrijf de tabel als een relatie door alle attribuuttypes op te sommen:

R0 (klantnummer, klantnaam, sectornummer, sectoromschrijving, artikelnummer, artikelomschrijving, aantalBesteld)

Duid in deze relatie alle herhalende gegevens aan. Dit zijn de attribuuttypes die in de bekomen tabel meerdere waarden per rij hebben (artikelnummer, artikelomschrijving, . . .). Zet deze per groep tussen haakjes met een sterretje erbij.

R0 (klantnummer, klantnaam, sectornummer, sectoromschrijving, (artikelnummer, artikelomschrijving, aantalBbesteld)*)

Opmerking: Het is mogelijk dat er meerdere herhalende groepen binnen een herhalende groep staan (genest). Deze noteren we dan simpelweg als volgt: (x, y, z, (a, b, c, (d, e, f)*)*).

Duid de sleutel aan door deze te onderlijnen. De sleutel is dat attribuut (of die combinatie van attributen) dat het hoogste niveau identificeert. Hier hebben we 2 niveaus: het niveau klant en per klant het niveau artikel. Het niveau klant wordt geïdentificeerd aan de hand van klantnummer (dat zie je aan de functionele afhankelijkheden).

R0 (**klantnummer**, klantnaam, sectornummer, sectoromschrijving,(artikelnummer, artikelomschrijving, aantalBesteld)*)

Nu hebben we een relatie in de nulde normaalvorm of ongenormaliseerde vorm.

Eerste Normaalvorm

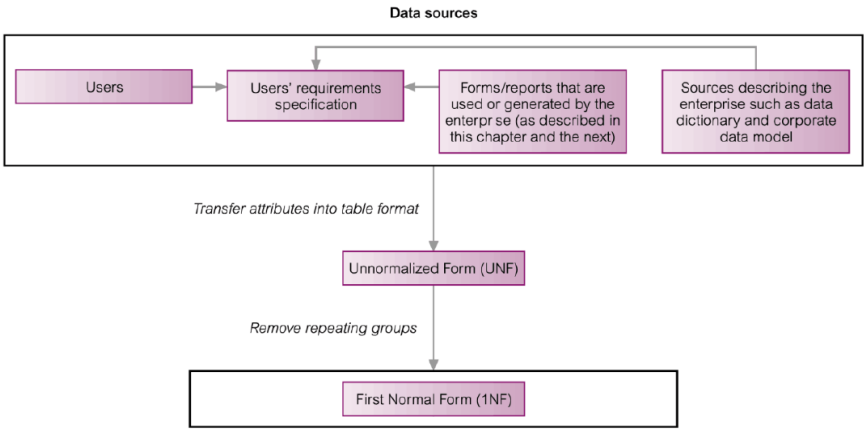
Een relatie in de eerste normaalvorm (6.13) moet aan enkele voorwaarden voldoen, namelijk:

- *geen afgeleide attribuuttypes*

  Een afgeleid attribuuttype is een procesgegeven zoals bijvoorbeeld leeftijd van een persoon. Dit kan je berekenen aan de hand van de geboortedatum.

- *geen samengestelde attribuuttypes*

  Een samengesteld attribuuttype is een gegeven dat nog kan opgesplitst worden. Een voorbeeld hiervan is adres dat is samengesteld uit straat, huisnummer, postcode, gemeente, land.



Figuur 6.13: Eerste Normaalvorm

- *geen meerwaardige attribuuttypes*

  Een meerwaardig attribuuttype kan meerdere waarden bevatten. Meerwaardige attribuuttypes hebben we in R0 reeds aangeduid met een * (herhalende gegevens). In ons voorbeeld worden artikelnummer, artikelomschrijving en aantalBesteld meerdere keren per bestelling herhaald.

De eerste twee voorwaarden zijn relatief eenvoudig en vergen dan ook geen verdere uitleg. Om aan de derde voorwaarde te voldoen moeten er weer enkele stappen doorlopen worden zodat de nulde normaalvorm omgezet kan worden naar een eerste normaalvorm.

Stappenplan eerste normaalvorm

1. Schrap alle herhalende groepen uit de nulde normaalvorm.

   R0 (**klantnummer**, klantnaam, sectornummer, sectoromschrijving, ~~(artikelnummer, artikelomschrijving, aantalBesteld)~~*)

2. Zonder elke herhalende groep af in een nieuwe relatie. De sleutel van de relatie waaruit de herhalende groep komt moet steeds meegenomen worden.

   R1.1 (**klantnummer**, klantnaam, sectornummer, sectoromschrijving)
   R1.2 (**klantnummer** , artikelnummer, artikelomschrijving, aantalBesteld)

3. Kies een sleutel voor de nieuwe relatie, baseer je hiervoor op de reeds bepaalde functionele afhankelijkheden.

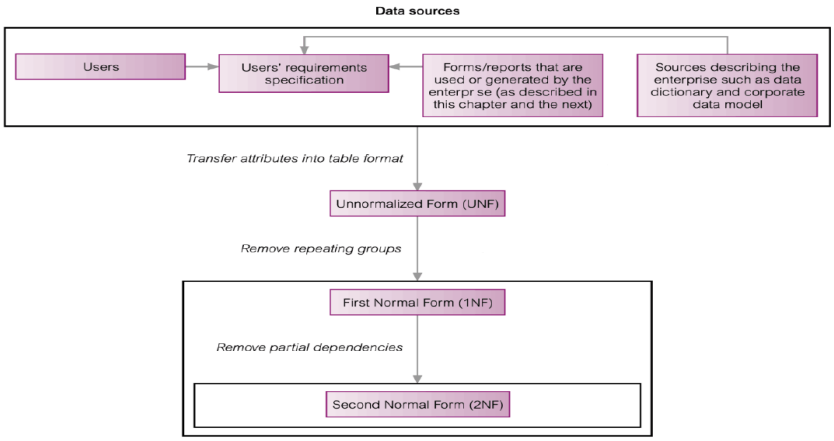
   R1.1 (**klantnummer**, klantnaam, sectornummer, sectoromschrijving)

   R1.2 (**klantnummer, artikelnummer** , artikelomschrijving, aantalBesteld)

Opmerking: Indien de afgezonderde herhalende groep eveneens een herhalende groep bevat dien je stappen 2 en 3 te herhalen totdat er geen herhalende gegevens meer zijn.

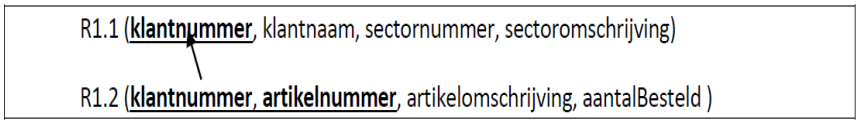
Tweede Normaalvorm

Een relatie in de tweede normaalvorm bevat geen partiële afhankelijkheden. An-



Figuur 6.14: Tweede Normaalvorm

ders gezegd alle attribuuttypes zijn afhankelijk van de volledige sleutel en niet van slechts een deeltje. Een relatie in de tweede normaalvorm moet natuurlijk nog altijd voldoen aan de voorwaarden van de eerste normaalvorm. Je vertrekt dus van de relaties uit de eerste normaalvorm en volgt dan de stappen hieronder beschreven om de partiële afhankelijkheden weg te werken.



Figuur 6.15: Voorbeeld Tweede Normaalvorm

Stappenplan tweede normaalvorm

1. Zoek in de bekomen relaties van de eerste normaalvorm niet-sleutelattribuuttypes die partieel afhankelijk zijn van de primaire sleutel.

   Vermits partiële afhankelijkheid enkel mogelijk is bij relaties met een samengestelde sleutel, hoeven we de relatie R1.1 niet verder te onderzoeken. R2.1 zal identiek zijn aan R1.1.
   R2.1 (**klantnummer**, klantnaam, sectornummer, sectoromschrijving)

   R1.2 heeft wel een samengestelde sleutel (klantnummer, artikelnummer) dus is er een kans op partiële afhankelijkheid. Onderzoek voor elk niet-sleutelattribuut of het functioneel afhankelijk is van een deel van de sleutel.

   Niet-sleutelattributen: artikelomschrijving en aantalBesteld.

   We bekijken voor beide attribuuttypes of ze partieel functioneel afhankelijk zijn:

   artikelomschrijving: wanneer we de reeds bepaalde functionele afhankelijkheden bekijken bemerken we dat een artikelomschrijving uniek bepaald kan worden aan de hand van een artikelnummer. Het attribuuttype artikelomschrijving is dus partieel afhankelijk van de sleutel (klantnummer, artikelnummer) in de relatie R1.2.

   aantalBesteld wordt bepaald door (klantnummer, artikelnummer) en is dus volledig functioneel afhankelijk van de sleutel.

2. Om de partiële afhankelijkheid weg te werken vormen we een nieuwe relatie bestaande uit het afhankelijke sleuteldeel + de niet-volledig afhankelijke attribuuttypes. Het afhankelijke sleuteldeel vormt steeds de sleutel van de nieuwe relatie. Als we dit toepassen op ons voorbeeld bekomen we volgende relatie

   R2.2 (**artikelnummer**, artikelomschrijving)

3. Verwijder de partieel afhankelijke attribuuttypes uit de oorspronkelijke relatie. Door dit toe te passen op de relatie R1.2 bekomen we:

   (**klantnummer , artikelnummer**, aantalBesteld)

4. Schrijf het nieuw model uit:

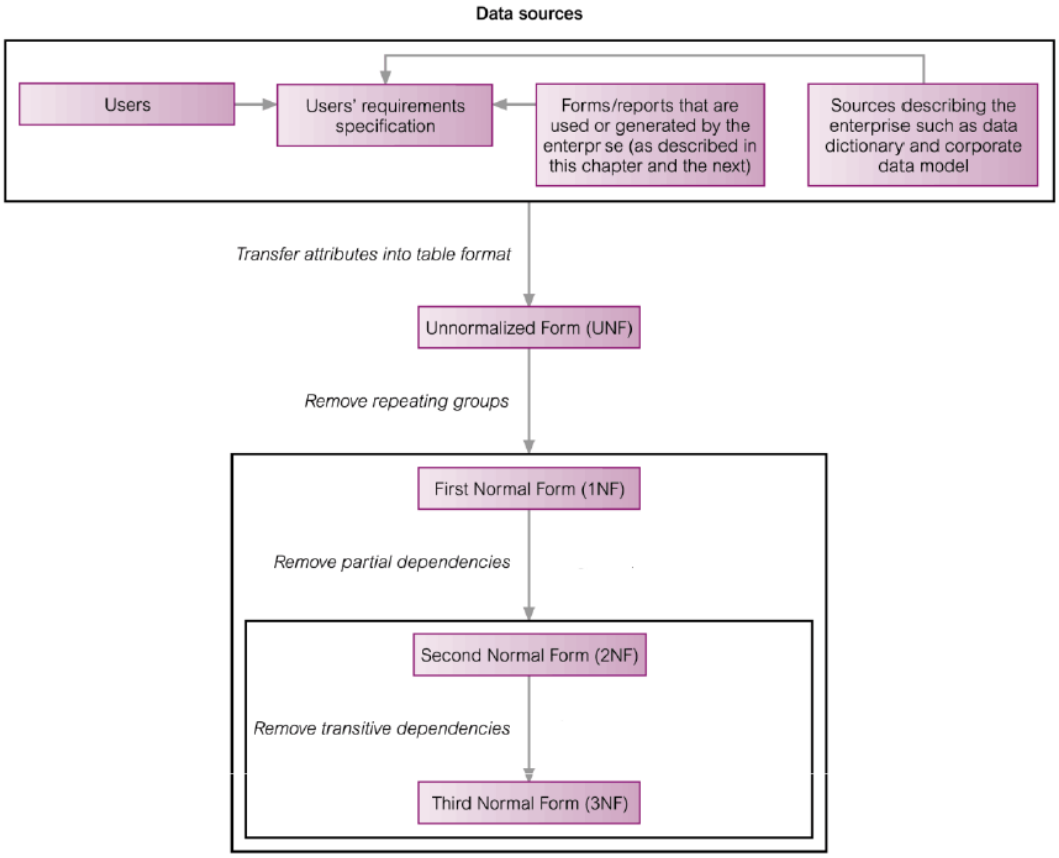
   R2.1 (**klantnummer**, klantnaam, sectornummer, sectoromschrijving)

R2.2 (**klantnummer** , **artikelnummer**, aantalBesteld)

R2.3 (**artikelnummer**, artikelomschrijving)

Derde Normaalvorm

Een relatie in de derde normaalvorm mag geen transitieve afhankelijkheden bevatten

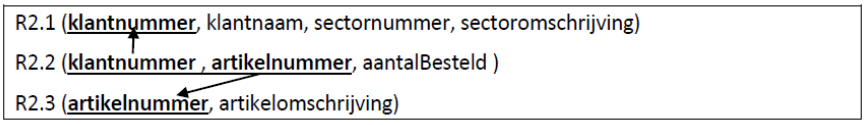


Figuur 6.16: Derde Normaalvorm

en moet nog steeds voldoen aan de voorwaarden van de eerste en tweede normaalvorm.

We vertrekken dus van de relaties uit de 2^e normaalvorm: zie figuur 6.17.

Om een relatie van de tweede normaalvorm om te zetten naar de derde dienen we elk voorkomen van transitieve afhankelijkheid te verwijderen.



Figuur 6.17: Voorbeeld tweede normaalvorm

Stappenplan derde normaalvorm

1. Zoek op basis van de reeds bepaalde functionele afhankelijkheden de attribuuttypes die transitief afhankelijk zijn van één of meerdere niet-sleutelattributen.

   In het door ons gebruikte voorbeeld bemerken we dat in de relatie R2.1 het attribuuttype sectoromschrijving afhankelijk is van het attribuuttype sectornummer en niet van de primaire sleutel (klantnummer).

2. Vorm een nieuwe relatie bestaande uit de transitieve attribuuttypes en het attribuuttype waarvan ze afhankelijk zijn. Dit attribuuttype vormt de sleutel van de nieuwe relatie.

   (**sectornummer**, sectoromschrijving)

3. Verwijder de transitief afhankelijke attribuuttypes uit de oorspronkelijke relatie

   (**klantnummer**, klantnaam, sectornummer)

4. Schrijf het nieuw model uit:

   R3.1 (**klantnummer**, klantnaam, *sectornummer*)

   R3.2 (**sectornummer**, sectoromschrijving)

   R3.2 (**klantnummer** , **artikelnummer**, aantalBesteld)

   R3.3 (**artikelnummer**, artikelomschrijving)

Het relationele model

Na het bepalen van de 3^{de} normaalvorm benoemen we de gevonden relaties met een betekenisvolle naam. We vergeten ook de integriteitregels niet bij elke vreemde sleutel:

Klant (**klantnummer**, klantnaam, sectornummer)

Vreemde sleutel sectornummer verwijst naar Sector.sectornummer, verplicht

Sector (**sectornummer**, sectoromschrijving)

Bestelling (**klantnummer , artikelnummer**, aantalBesteld)

Vreemde sleutel klantnummer verwijst naar Klant.klantnummer, verplicht

Vreemde sleutel artikelnummer verwijst naar Artikel.artikelnummer, verplicht

Artikel (**artikelnummer**, artikelomschrijving)

Uitgewerkt voorbeeld normalisatie

We zullen in dit voorbeeld 3 documenten normaliseren en telkens het bekomen resultaat integreren in het relationeel model.

Op alle documenten (zie 6.18) is de informatie over de klant steeds aanwezig.

Na normalisatie van het eerste document (zie figuur 6.19) bekomen we het relationeel model zoals te zien in figuur: 6.20.

Na normalisatie van het tweede document (zie figuur 6.21) bekomen we het relationeel model zoals te zien in figuur: 6.22.

Integratie van beide relationele modellen bezorgt ons het tussentijds relationeel model zoals te zien in figuur 6.23.

Na normalisatie van het derde document (zie figuur 6.24) bekomen we het relationeel model zoals te zien in figuur: 6.25.

Finale integratie De finale integratie van de 3 documenten levert ons uiteindelijk een relationeel model met 6 tabellen. Zie figuur: 6.26

Opmerking: Geladen artikelen is overbodig: het aantal dat geladen is, is het aantal dat geleverd is (Geleverde artikelen).

Daarentegen is het aantal bij geleverde artikelen geen procesgegeven!! Het aantal besteld is niet altijd gelijk aan het aantal geleverd!

Document 1: Bestelbon

Bestelbonnummer:	2345	Datum:	22-8-2006
Klant:	2087 Janssens bvba	Bredabaan 234	2060 Merksem

Artcode	Omschrijving	Aantal
A1234	Laserprinter HP 1455	25
B2345	PC Packard Bell XP/500G	20
A0255	Inkjetprinter HP 7660	15
B1346	Laptop Toshiba Satellite	30

Document 2: Zendnota

Zendnotanummer	1234	Datum:	25-8-2006
Klant:	2087 Janssens bvba	Bredabaan 234	2060 Merksem

Artcode	Omschrijving	Aantal	Nr bon	Datum bon	Nog te leveren
A1234	Laserprinter HP 1455	25	2345	22-8-2006	
B2345	PC Packard Bell XP/500G	15	2173	18-8-2006	
		10	2345	22-8-2006	10
B1346	Laptop Toshiba Satellite	20	2137	18-8-2006	
		30	2345	22-8-2006	

Document 3: Picking ticket

			Datum:	25/08/2006
Klant:	2087	Janssens bvba	Bredabaan 234	2060 Merksem

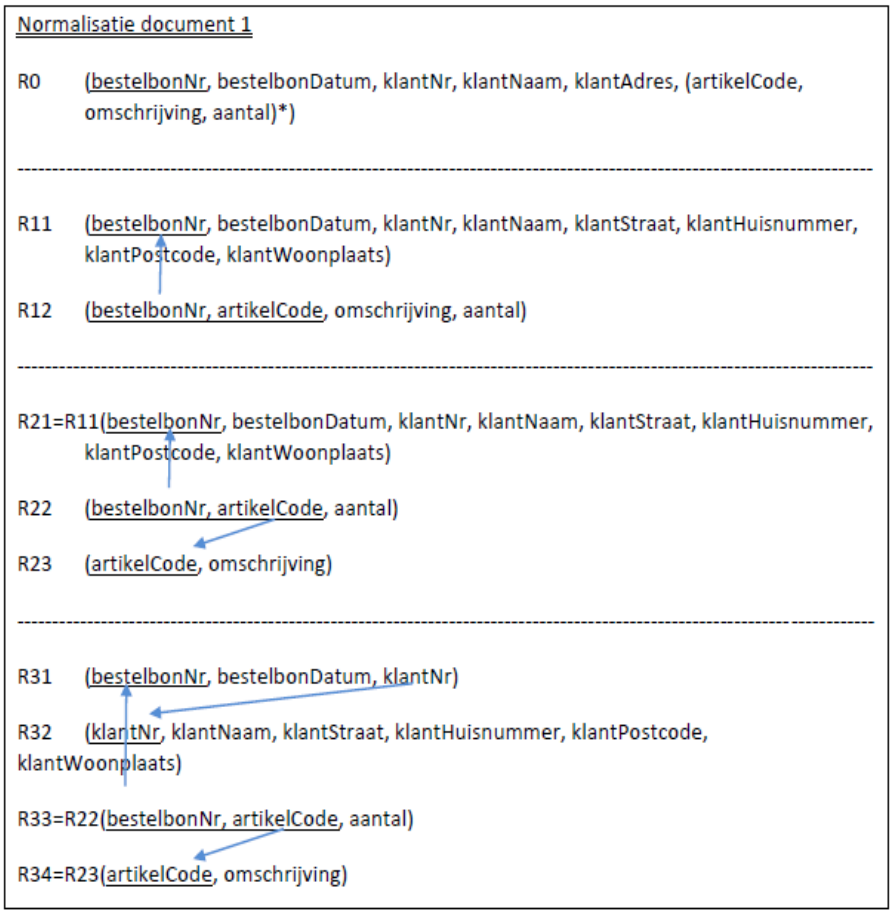
Rij	Rek	Artcode	Omschrijving	Aantal
3	5C	A1234	Laserprinter HP 1455	25
4	3B	B2345	PC Packard Bell XP/500G	25
5	1A	B1346	Laptop Toshiba Satellite	50

Figuur 6.18: Drie documenten ter normalisering

6.3 Valideren van relaties door normalisatie

De relaties, verkregen uit het mappen van het conceptuele model naar het relationele model, moeten gecontroleerd worden of ze in de derde normaalvorm staan.

Dit doe je door te controleren of geen enkele voorwaarde van de 3^{de} normaalvorm geschonden wordt. Wanneer dit niet zo is kan je bijna zeker zijn dat er geen update, insert of delete anomalieën zullen optreden.



Figuur 6.19: Normalisatie eerste document

Figuur 6.20: Relationeel model van het eerste document

Figuur 6.21: Normalisatie tweede document

6.4 Valideren of de relaties voldoen aan de gebruikersvoorwaarden

In deze stap wordt gecontroleerd of het model voldoet aan alle eisen van de gebruikers. Met andere woorden, we moeten controleren of alle transacties die de gebruikers willen uitvoeren op de gegevens mogelijk zijn.

Bekijk het doel waarvoor de databank gebruikt moet worden en controleer bij elke informatiebehoefte of deze informatie inderdaad uit de databank afgeleid kan worden.

Relationeel model

Zendnota(zendnotaNummer, zendnotaDatum, klantNummer)
VS klantNr verwijst naar Klant.klantNr, verplicht

Klant (klantNummer, klantNaam, klantStraat, klantHuisnummer, klantPostcode, klantWoonplaats)

Artikel (artikelCode, artikelOmschrijving)

Geleverde artikelen(zendnotaNummer, artikelCode, bonNr, aantal)
VS zendnotaNummer verwijst naar Zendnota.zendnotaNummer, verplicht
VS artikelCode verwijst naar Artikel.artikelCode, verplicht
VS bonNr verwijst naar Bon.bonNr, verplicht

Bon (bonNr, bonDatum)

Figuur 6.22: Relationeel model van het tweede document

Integratie van beide documenten

Bestelbon (bestelbonNr, bestelbonDatum, klantNr)
VS klantNr verwijst naar Klant.klantNr, verplicht

Klant (klantNr, klantNaam, klantStraat, klantHuisnummer, klantPostcode, klantWoonplaats)

Bestelde artikelen(bestelbonNr, artikelCode, aantal)
VS bestelbonNr verwijst naar Bestelbon.bestelbonNr, verplicht
VS artikelCode verwijst naar Artikel.artikelCode, verplicht

Artikel (artikelCode, omschrijving)

Zendnota(zendnotaNummer, zendnotaDatum, klantNummer)
VS klantNummer verwijst naar Klant.klantNr, verplicht

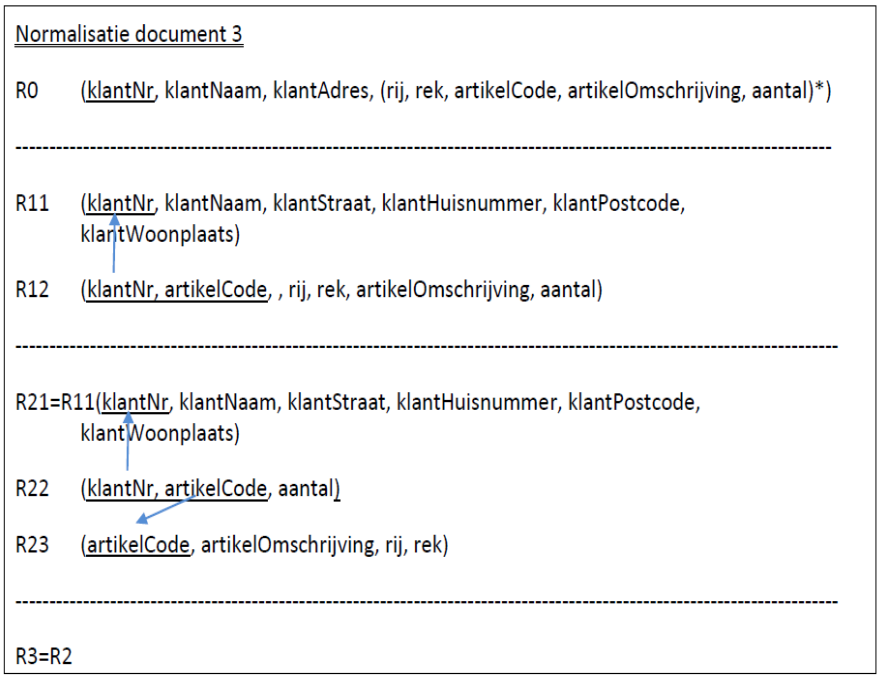
Geleverde artikelen(zendnotaNummer, artikelCode, bonNr, aantal)
VS zendnotaNummer verwijst naar Zendnota.zendnotaNummer, verplicht
VS artikelCode verwijst naar Artikel.artikelCode, verplicht
VS bonNr verwijst naar Bestelbon.bestelbonNr, verplicht

Figuur 6.23: Integratie van beide documenten

6.5 Beperkingen van het relationele model

Het relationele model bevat enkele structuurbeperkingen die te maken hebben met het feit dat er bijna altijd 1-op-veel relaties voorgesteld worden. Daardoor raakt veel semantiek verloren uit het conceptuele model.

Relaties worden gelegd aan de hand van vreemde sleutels. De vreemde sleutel is bijna nooit uniek en verwijst naar de primaire sleutel van een andere tabel. Daardoor



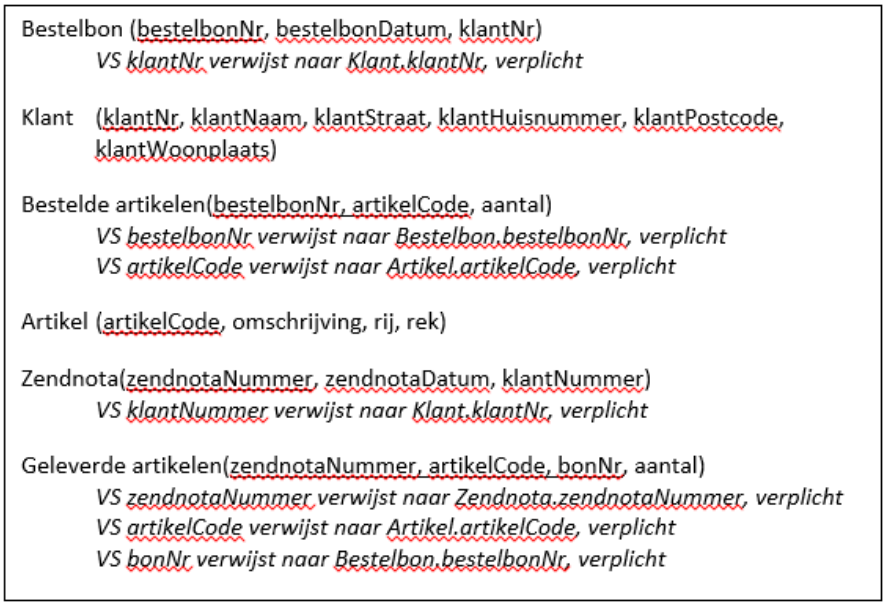
Figuur 6.24: Normalisatie derde document

Figuur 6.25: Relationeel model van het derde document

heb je steeds 1-op-veel relaties. De enige manier om een 1-op-1 verband te behouden is het leggen van een relatie tussen een primaire sleutel en een vreemde sleutel die beide uniek zijn. Dit komt bijvoorbeeld voor als resultaat van een mapping van een 'optional'-specialisatie.

Ook minimumcardinaliteit 1 bij een maximumcardinaliteit N raakt verloren in de mapping.

Voorbeeld 1: 1-op-1 relatie die behouden blijft:



Figuur 6.26: Relationeel model door finale integratie van de drie documenten

Persoon (persoonNummer, voornaam, familienaam)

Klant (persoonNummer, straat, huisnr, postcode, woonplaats)

VS persoonNummer verwijst naar Persoon.persoonNummer, verplicht

Personeelslid (persoonNummer, datumInDienst, loon)

VS persoonNummer verwijst naar Persoon.persoonNummer, verplicht

Voorbeeld 2: 1-op-1 relatie die een 1-op-veel wordt:

Personeelslid (personeelsNummer, voornaam, familienaam)

Bedrijfswagen (nummerplaat, merk, type, personeelslid)

VS personeelslid verwijst naar Personeelslid.personeelsnummer, verplicht

Een bedrijfswagen is van maximum 1 personeelslid: is voldaan.

Een personeelslid kan maar van 1 auto hebben: kan niet gegarandeerd worden in dit relationeel model! Het is in dit model perfect mogelijk dat meerdere tupels van auto verwijzen naar hetzelfde personeelslid.

Oplossing: in relationeel model een beperkingsregel toevoegen:

Personeelslid (personeelsNummer, voornaam, familienaam)

Bedrijfswagen (nummerplaat, merk, type, personeelslid)

VS personeelslid verwijst naar Personeelslid.personeelsnummer, verplicht, uniek

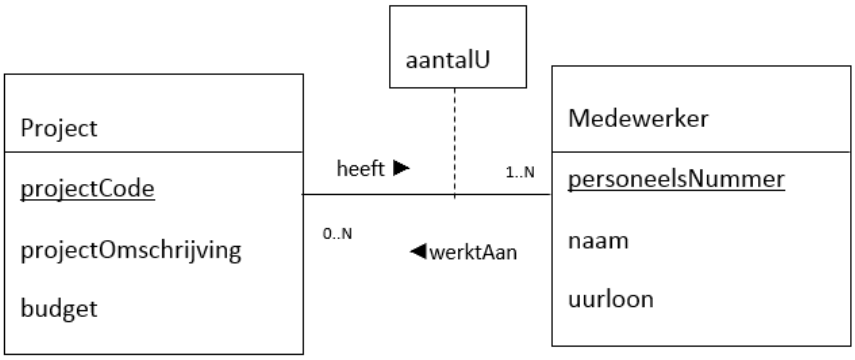
Opmerking: deze beperkingsregel kan enkel maar toegevoegd worden indien er een verplichte deelname is!! Indien NULL-waarden toegelaten zijn voor de vreemde sleutel kan je uiteraard niet stellen dat deze uniek moet zijn.

Voorbeeld 3: minimumcardinaliteit 1 bij maximumcardinaliteit N verdwijnt:

Een project heeft meerdere medewerkers (minstens 1); een medewerker kan aan meerdere projecten werken. Er wordt per medewerker, per project het aantal uur bijgehouden dat deze medewerker aan dit project werkt.

Het conceptueel model zien we in figuur 6.27.

Na modellering krijgen we volgend relationeel model:



Figuur 6.27: Conceptueel model

Project (projectCode, projectOmschrijving, budget)

Medewerker (personeelsNummer, voornaam, familienaam, uurloon)

Project/Medewerker (projectCode, personeelsNummer, aantalU)

VS projectCode verwijst naar Project.projectCode, verplicht

VS personeelsNummer verwijst naar medewerker.personeelsNummer, verplicht

Het relationeel model biedt geen garantie dat elk project minstens 1 medewerker heeft!

Dit kan opgelost worden door het gebruik van triggers. Deze worden gedefinieerd in het fysiek model. (zie cursus tweede jaar)

Deel II

SQL

Inleiding

De ontwikkeling van SQL (Structured Query Language) is begonnen in het midden van de jaren 1970 op het IBM onderzoekssinstituut San José. De eerste versie heette SEQUEL. Na verschillende versies van SEQUEL werd de taal in 1980 tot SQL gedoopt.

Sindsdien hebben ook tal van andere leveranciers SQL-producten ontwikkeld: Oracle, SyBase, SQL Server, Microsoft Access, ...

Het ANSI (American National Standards Institute) heeft deze taal als standaard gekozen voor het manipuleren van gegevens uit databanken. Bepaalde implementaties van SQL (b.v.b. in ORACLE of SQL Server) kunnen afwijken van de ANSI standaard. Dit is o.a. te wijten aan het feit dat veel DBMS producten al waren ontwikkeld voordat de standaard werd opgesteld.

Tijdens de bespreking van SQL gebruiken we de syntax van MySQL. In de voorbeelden maken we gebruik van de tabellen uit appendix A.

Te verwerven competenties

- Kan eenvoudige queries uitvoeren via SQL (SELECT FROM WHERE GROUP BY HAVING).
- Kan de data in een databank bijwerken via SQL (INSERT UPDATE DELETE).
- Kan de structuur van een databank bijwerken via SQL (CREATE - ALTER - DROP).

HOOFDSTUK 7

Werken met één tabel

7.1 Algemeen

Allereerst gaan we na wat SQL biedt voor het bewerken van een afzonderlijke tabel. SQL statements zijn niet case-sensitive. Je kan een SQL statement over meerdere lijnen uitsplitsen of laten inspringen. Dit bevordert de leesbaarheid.

7.2 Alle rijen en kolommen

Om alle gegevens uit een tabel te tonen gebruiken we * als parameter in de SELECT-clausule.

```
SELECT *
FROM Werknemer
```

Opmerking : De standaardwoorden SELECT en FROM zijn verplicht. De gewenste kolommen worden na de SELECT genoemd. * is een verkorte notatie om niet alle namen van de kolommen te moeten opgeven (hier zou men 12 kolomnamen moeten opgeven). De tabel waaruit geselecteerd moet worden, wordt genoemd na de FROM.

7.3 Subset van rijen (SELECTIE)

Wensen we slechts een subset van de tabel te zien, dan vermelden we in WHERE-clausule de voorwaarden waaraan de getoonde records moeten voldoen.

```
SELECT *
FROM Werknemer
WHERE AfdNr ='D11'
```

Opmerking : In de WHERE-clausule worden de selectievoorwaarden opgegeven.

7.4 Subset van kolommen (PROJECTIE)

Indien we niet alle, maar slechts een beperkt aantal kolommen wensen te zien, vermelden we de kolomnamen in de SELECT-clausule.

```
SELECT Nr, FNaam, Afd
FROM Werknemer
```

7.5 Subset van rijen en kolommen

```
SELECT Nr, FNaam, Afd
FROM Werknemer
WHERE Afd = 'D11'
```

Opmerking: Vraag enkel die kolommen en rijen op die je nodig hebt:

- hoe meer velden,... hoe meer geheugen nodig is om de data op te slaan, hoe trager de applicatie;
- in client/server omgevingen vereist een grotere recordset meer netwerk traffic.

7.6 Voorwaarden aan rijen opleggen

7.6.1 Vergelijkingsoperatoren

In de zoekvoorwaarde kan je gebruik maken van de relationele operatoren: $=$, $<>$,$>$, $>=$, $<$, $<=$ voor het vergelijken van numerische, alfanumerische waarden en datums.

Voorbeeld: Selecteer het nummer, de voornaam en de familienaam van de werknemers met een jobcode 56

```
SELECT NR, VNaam, FNaam
FROM Werknemer
WHERE Code = 56
```

7.6.2 Gebruik van LIKE of NOT LIKE

Hierdoor kan men karakterstrings opsporen die gedeeltelijk overeenstemmen met een opgegeven string.
In deze opgegeven string stelt een %-teken om het even welke string van 0 of meer karakters voor.

Voorbeeld:

- naam beginnend met P: 'P%'
- naam eindigend op C: '%C'
- naam beginnend met ABC: 'ABC%'
- naam eindigend met XYZ: 'XYZ%'
- naam beginnend met A en eindigend op C: 'A%C'
- naam die ergens SON bevat: '%SON%'

Een '_' teken stelt in een string 1 enkel karakter voor.

Voorbeeld:

- naam van 5 karakters beginnend met P: 'P____'
- telefoonnummer van 5 cijfers met op middelste plaats een 8: '__8__'

Opmerking: indien je de wildcard-symbolen % of _ wenst op te sporen in een string moet je deze voorafgaan door een \. Zo vind je met 'A_%' alle strings beginnend met A gevolgd door een _.

Opgave: Geef het nummer, de naam en het afdelingsnummer van alle werknemers, waarvan de familienaam start met een P en die in een afdeling werken beginnend met D en als 3^{de} karakter een 1 hebben.

```
SELECT Nr, VNaam, FNaam, Afd
FROM Werknemer
WHERE FNaam LIKE 'P%' AND Afd LIKE 'D_1'
```

7.6.3 AND, OR en NOT

SQL evalueert eerst de NOT, dan de AND en dan de OR. Je kan de verwerkingsvolgorde beïnvloeden door haakjes te gebruiken. Vermijd het gebruik van de NOT operator, dit vereist het doorzoeken van alle rijen uit de tabel.

Enkele voorbeelden:

- Selecteer het nummer, de voornaam en de familienaam van de werknemers met een jobcode 56 uit de afdeling D11

```
SELECT Nr, VNaam, FNaam
FROM Werknemer
 WHERE Afd = 'D11' AND Code = 56
```

- Selecteer werknemers die ofwel in afdeling D11 of in D21 werken en met een code groter dan 54 of een opleidingsniveau hoger dan 15.

```
SELECT Nr, VNaam, FNaam
FROM Werknemer
WHERE (Afd = 'D11' OR Afd = 'D21') AND ( Code > 54 OR Niv > 15)
```

- Geef voornaam en familienaam van de werknemers met code 54, die in een willekeurige afdeling werken met uitsluiting van afdeling D11.
  - 1^{ste} versie: a.d.h.v. de NOT operator

```
SELECT VNaam, FNaam, Afd, Niv
FROM Werknemer
WHERE Code = 54 AND NOT ( Afd = 'D11')
```

  - 2^{de} versie met de <> operator

```
SELECT VNaam, FNaam, Afd, Niv
FROM Werknemer
WHERE Code = 54 AND (Afd <> 'D11')
```

7.6.4 BETWEEN

BETWEEN of NOT BETWEEN wordt gebruikt bij selectie van een veld binnen een bepaald waardeverloop.

Voorbeeld: Geef het nummer, de naam en het afdelingsnummer van alle werknemers met een nummer tussen 100 en 230.

```
SELECT Nr, VNaam, Fnaam, Afd
FROM Werknemer
WHERE Nr BETWEEN '100' AND '230'
```

Nota: BETWEEN impliceert een gesloten interval, d.w.z. inclusief de beide grenzen van het interval.

7.6.5 IN

Hierdoor kan men de inhoud van een veld vergelijken met een lijst van waarden (uiteraard is ook NOT IN mogelijk).

Voorbeeld: Geef het nummer, de naam en het opleidingsniveau van werknemers met niveau 16, 18 of 20.

```
SELECT Nr, VNaam, Fnaam, Niv
FROM Werknemer
WHERE Niv IN (16, 18, 20)
```

7.6.6 Gebruik van NULL-waarden

NULL is een speciale waarde (verwar niet met blanco of met 0 (zero)). NULL wil zeggen: ofwel bestaat de informatie niet (b.v.b. persoon heeft geen telefoonnummer) ofwel is de informatie nog onbekend op het moment van opname in de DB-tabel (b.v.b. datum van uitdiensttreding). NULL waarden komen voor wanneer er bij input in een bepaalde kolom geen waarde werd ingebracht en er voor deze kolom geen default waarden voorzien zijn.
Bijvoorbeeld bij de indienstneming van nieuwe Werknemers kan het zijn dat bij stockering in de tabel Werknemer enkel nummer, naam, werkafdeling, datum indiensttreding en geslacht bekend zijn. Alle andere velden zullen dan automatisch met NULL worden opgevuld.

Opgave: Selecteer alle Werknemers met als jobcode NULL

```
SELECT Nr, VNaam, FNaam
FROM Werknemer
WHERE Code IS (*)NULL
```

(*)Hier moet men schrijven IS, dus geen =, want met NULL is geen enkele vergelijking mogelijk.

7.6.7 Gecombineerde predicaten

We kunnen al het vorige combineren in 1 WHERE-clausule.

Voorbeeld:

```
SELECT FNaam, Code, Niv, Salaris, Afd
FROM Werknemer
WHERE ( Afd = 'D11' OR Adf = 'E21')
      AND Niv IN (12, 14, 16, 18)
      AND Salaris BETWEEN 15600 AND 23700
      AND (FNaam NOT LIKE 'P%' OR FNaam LIKE '%DE%')
      AND Code IS NOT NULL
```

7.7 Formatteren van de resultaten

Je kan het uitzicht van de resultset, bekomen als resultaat van een query, aanpassen door te sorteren, andere benamingen te geven aan de kolommen, duplicaten te elimineren of zelf bepaalde resultaten te berekenen.

7.7.1 Gebruik van ORDER BY

Het resultaat van een SELECT-bevel wordt voorgesteld in een volgorde bepaald door het systeem. Door de ORDER BY-clausule kan de gebruiker zelf specifiëren, hoe het

resultaat geordend moet zijn.

Algemene vorm:

ORDER BY *kolom-specificatie [ASC/DESC] [,kolom-specificatie[ASC/DESC]]*

Er kan dus ordening gebeuren volgens meer dan 1 criterium. De gegevens worden dan eerst gesorteerd op de eerstgenoemde kolom, daarbinnen op de tweede, enz.
Vermeld je niets na de kolom-specificatie, dan is het opklimmende orde (ascending: ASC). Dit is meteen de DEFAULT-optie. Voor afdalende orde moet je na de kolom-specificatie het woord DESC vermelden.
Kolom-specificatie is ofwel een naam van een kolom, of een nummer. Namen kunnen enkel gebruikt worden als het over welbepaalde kolommen gaat. Naar expressies refereereer je d.m.v. nummers (tenzij je aliassen gebruikt, zie later).

Voorbeeld: Geef alle Werknemers uit de afdelingen A00, B01, C01 en D01, geordend volgens afdelingsnummer. Binnen een afdeling moeten de rijen geordend zijn in afdalende volgorde van het maandelijks salaris.

```
SELECT FNaam, Afd, Salaris/12
FROM Werknemer
WHERE Afd = 'A00' OR Afd = 'B01' OR Afd = 'C01' OR Afd = 'D01'
ORDER BY Afd, 3 DESC
```

Uitleg: naar het 3^{de} item uit de SELECT kan je niet refereren d.m.v. een naam, want het is een expressie.

7.7.2 Gebruik van DISTINCT/ALL

Het resultaat van een SELECT kan een lijst opleveren waarin bepaalde rijen meer dan één keer voorkomen. SQL verwijdert niet automatisch dubbele rijen omdat dit veel tijd kost en omdat dit vaak ook niet nodig of wenselijk is. Als dubbele rijen wel moeten worden verwijderd, moet het SQL-woord DISTINCT als positioneel sleutelwoord, onmiddellijk na SELECT, worden toegevoegd. Met het SQL-woord ALL (ALL is de default-optie) worden ook dubbele rijen weergegeven.

Voorbeelden:

- SELECT FNaam (of SELECT ALL FNaam)
  FROM Werknemer
  WHERE FNaam = 'De Bruyn'
- SELECT DISTINCT FNaam
  FROM Werknemer
  WHERE FNaam = 'De Bruyn'

Stel dat er b.v.b. in de tabel Werknemer 5 rijen zijn met familienaam 'De Bruyn' dan zal in het 1^{ste} voorbeeld 5 keer 'De Bruyn' onder elkaar verschijnen. In het 2^{de} voorbeeld zal in dergelijk geval de naam 'De Bruyn' maar 1 keer verschijnen.

Oplossing:

```
SELECT DISTINCT VNaam, FNaam
FROM Werknemer
WHERE FNaam= 'De Bruyn'
```

Hier worden alleen rijen uitgeschakeld met dezelfde voornaam én familienaam. We kunnen echter wel rijen bekomen met 'De Bruyn' als familienaam, maar met telkens een verschillende voornaam.

7.7.3 Gebruik van aliassen (leesbare namen) voor kolommen

Per default worden de kolomnamen, zoals gedefinieerd bij creatie van een tabel, gebruikt. Soms zijn deze namen nogal cryptisch of van weinig betekenis voor de gebruiker. Je kan de naam van een kolom aanpassen a.d.h.v. kolomaliassen. De aliassen kan je ook gebruiken in de ORDER BY clause.

Voorbeeld:
```
SELECT DISTINCT VNaam AS 'Voornaam', FNaam AS 'FamilieNaam'
FROM Werknemer
WHERE FNaam= 'De Bruyn'
```

7.7.4 Berekende resultaatkolommen

Je kan ook berekeningen of formatteringen toepassen op kolommen.

- Zo zou je b.v.b. het salaris op jaarbasis kunnen weergeven i.p.v. het maandsalaris.

  ```
  SELECT VNaam, FNaam, Salaris*12 as 'JaarSalaris'
  FROM Werknemer
  ```

- Je zou de naam en voornaam van een werknemer ook in 1 kolom kunnen weergeven. In dit voorbeeld worden naam en voornaam, jaarsalaris getoond van alle werknemers waarvoor jaarsalaris $>$ 12.000 euro en dit gesorteerd op jaarsalaris.

  ```
  SELECT VNaam + " " + FNaam as 'Werknemer', Salaris*12 as 'JaarSalaris'
  FROM Werknemer
  WHERE (salaris * 12) > 12000
  ORDER BY jaarsalaris
  ```

7.7.5 Enkele handige functies in MySQL

MySQL beschikt over een uitgebreide bibliotheek van ingebouwde functies. Hieronder worden er een paar opgesomd.

- Datum- en tijdfuncties:
  - CURDATE(): geeft de huidige datum in formaat 'JJJJ-MM-DD'.
  - CURTIME(): geeft het huidig tijdstip in formaat 'UU-MM-SS'.
  - DATEDIFF(expr1, expr2): geeft het verschil expr1 - expr2 in aantal dagen.
  - DATE_FORMAT(datum, format): geeft de datum weer volgens de formaat-specificaties in format. (zie Reference Manual voor alle formaatspecificaties). Voorbeeld: SELECT DATE_FORMAT('2017-10-15', '%W %d %M %Y') geeft Sunday 15 October 2017.
  - DAY(): is synoniem voor DAYOFMONTH() en geeft de dag van de maand weer als getal tussen 1 en 31.
  - DAYOFWEEK(datum): geeft het dagnummer in de week van de datum (1=zondag, . . . , 7=zaterdag).
  - NOW(): geeft de huidige datum en tijd in formaat 'YYYY-MM-DD HH:MM:SS'.
- Stringfuncties:
  - CONCAT(str1, str2, . . .): geeft een string weer bestaande uit de concatenatie van alle argumenten.
    Voorbeeld: SELECT CONCAT('My', 'S', 'QL', ' versie ', 2) geeft 'MySQL versie 2'.
  - LEFT(str, len): geeft een string bestaande uit de eerste *len*-tekens van *str*.
    Voorbeeld: SELECT LEFT('FOODBAR', 3) geeft 'FOO'.
  - LENGTH(str): geeft de lengte van de string *str* in bytes.
    Voorbeeld: SELECT LENGTH(concat('My', 'S', 'QL', ' versie ', 2)) geeft 14.
  - MID(str,pos, len) is een synoniem voor SUBSTRING(str,pos,len) en geeft de substring van *str* vanaf positie *pos* met lengte *len*.
    Voorbeeld: SELECT MID('FOODBAR', 3, 2) geeft 'OD'.
  - RIGHT(str, len): geeft de laatste *len* karakters van de string *str*.
  - STRCMP(str1,str2): geeft 0 indien beide gelijk zijn, -1 indien de eerste string kleiner is dan de tweede volgens de gehanteerde sorteringscode, zoniet is het resultaat +1.
    Voorbeeld:
    SELECT STRCMP('schip', 'schip') geeft 0;
    SELECT STRCMP('schep', 'schip') geeft -1,
    SELECT STRCMP('schop', 'schip') geeft 1.
    Er wordt geen onderscheid gemaakt tussen hoofd- en kleine letters. Zo is ook SELECT STRCMP('SCHIP', 'schip') = 0.
- Control flow functies :

  - CASE WHEN [booleaanse expressie] THEN *result* [WHEN [booleaanse expressie] THEN *result*] [ELSE *result*] END
    Voorbeeld: SELECT
    CASE
    WHEN price IS NULL THEN 'Not yet priced'
    WHEN price < 10 THEN 'Very Reasonable Title'
    WHEN price >= 10 and price < 20 THEN 'Coffee Table Title'
    ELSE 'Expensive book!'
    END AS "Price Category", CONVERT(varchar(20), title) AS "Shortened Title"
    FROM pubs.dbo.titles
    ORDER BY price
  - IF(expr1,expr2,expr3): geeft expr2 indien expr1 waar is (en verschillend is van NULL), zoniet is het resultaat expr3.
    Voorbeeld: SELECT IF('aap'>'beer', 'aap', 'beer') geeft 'beer'.
  - IFNULL(expr1, expr2): geeft expr2 indien expr1 NULL is, zoniet geeft het expr1.
    Voorbeeld: SELECT IFNULL(1/0, 'ja') geeft 'ja'.
  - NULLIF(expr1,expr2) geeft NULL indien expr1 = expr2, zoniet geeft het expr1.

- Rekenkundige functies :
  - CEIL(x): synoniem van CEILING(x) geeft het kleinst geheel getal niet kleiner dan x.
    Voorbeeld: CEIL(-2.35) geeft -2, CEIL(2.35) geeft 3.
  - FLOOR(x): geeft het grootste geheel getal niet groter dan x.
  - POWER(x,y): synoniem voor POW(x,y) geeft x tot de macht y.
  - ROUND(x): geeft het geheel getal afgerond volgens de klassieke regels.

- Conversiefuncties :
  - CAST(expr AS type)
  - CONVERT(expr, type)
    Beide functies zetten de expr om volgens het gespecifieerde type.
    Voorbeelden:
    SELECT CAST(now() AS DATE) geeft 2017-10-15.
    SELECT CONVERT(3.14, CHAR) geeft '3.14'.

7.8 Statistische functies (Aggregate functions in SQL)

Er zijn 5 ingebouwde statistische functies, die in een SELECT kunnen voorkomen. Statistische functies houden geen rekening met NULL waarden, met uitzondering van de COUNT(*).

- **AVG**: berekent het gemiddelde van de elementen van een numerieke kolom (null-waarden worden verwaarloosd).
  Bij AVG (DISTINCT item) tellen de duplicaten niet mee.
- **SUM**: berekent de som van de elementen van een numerieke kolom (null elementen tellen niet mee).
  Bij SUM (DISTINCT item) tellen duplicaten niet mee.
- **MIN, MAX**: minimum of maximum van een kolom van eender welk type. Min en max hoeven niet noodzakelijk op numerieke kolommen te werken.
- **COUNT**: werkt zowel op de volledige tabel, als op een aparte kolom en bepaalt het aantal rijen van de tabel of het aantal velden in de desbetreffende kolom.
  - COUNT(*): telt het totaal aantal rijen (onverschillig het feit dat rijen null-waarden bevatten).
  - COUNT(kolomnaam): telt het aantal velden in een kolom, onverschillig de duplicaten, maar null-waarden tellen niet mee.
  - COUNT(DISTINCT kolomnaam): telt het aantal verschillende velden in een kolom, zonder null-waarden.

Opgave : Tel het aantal Werknemers uit de afdeling D11 en geef het maximum, het minimum en het gemiddeld salaris voor deze afdeling, alsook het aantal verschillende jobcodes uit deze afdeling. Geef ook de som van alle lonen betaald in D11.

```
SELECT COUNT(*), AVG(Salaris), MIN(Salaris), MAX(Salaris), COUNT(DISTINCT
Code), SUM(Salaris)
FROM Werknemer
WHERE Afd = 'D11'
```

Opmerkingen:

1. Indien er functies voorkomen in de select-clausule, moeten volgens de ANSI-standaard, alle items van de SELECT-lijst, als argument van één of andere functie optreden, behalve indien het item ook in een GROUP BY voorkomt (zie verder).
   Voorbeeld: Willen we in voorgaande vraag ook het afdelingsnummer afgedrukt zien, dan is volgend antwoord, volgens de ANSI-standaard, fout.

   ```
   SELECT Afd, COUNT(*), AVG(Salaris), MIN(Salaris), MAX(Salaris),COUNT(DISTINCT
   Code), SUM(Salaris)
   ```

```
FROM Werknemer
WHERE Afd = 'D11'
```

Correctie: vermits er built-in functies in de SELECT-clausule voorkomen, moeten alle kolommen nu argument zijn van een functie. Schrijf daarom in de oplossing: MIN(Afd) of MAX(Afd).

Bemerking: in MySQL krijg je hierbij evenwel geen foutmelding en wijkt MySQL dus af van de standaard. Het resultaat is evenwel niet altijd correct!

2. Built-in functies mogen niet voorkomen in de WHERE-clausule ook niet in de GROUP BY-clausule.
3. Built-in functies kunnen enkel nog voorkomen in de HAVING-clausule (zie verder).

7.9 Gebruik van GROUP BY

De standaardfuncties kunnen ook worden toegepast op bepaalde groepen van rijen uit een tabel. Dergelijke groepen worden gevormd door rijen te selecteren die dezelfde waarde hebben in 1 of meer kolommen. We kunnen b.v.b. de werknemers groeperen per afdeling. De GROUP BY-clausule wordt gebruikt om de rijen van een tabel te verdelen in groepen.
Het is echter altijd zo, dat men dan ingebouwde functies loslaat op deze afzonderlijke groepen. Een SELECT waarin zo'n GROUPING voorkomt, levert dan als resultaat 1 rij per groep op. Het is dan ook onmogelijk en zelfs zinloos om nog kolomnamen in zo'n SELECT te vermelden, tenzij deze vermeld wordt in de GROUPING zelf! Alle overige kolomnamen zullen uitsluitend optreden als argument van de ingebouwde functies.

Voorbeeld: Geef het maximum, het minimum en het gemiddeld salaris van alle Werknemers per afdeling.

```
SELECT Afd, MAX(Salaris), MIN(Salaris), AVG(Salaris)
FROM Werknemer
GROUP BY Afd
```

De rijen van de tabel Werknemer worden eerst gegroepeerd per afdeling en vervolgens worden maximum, minimum en gemiddeld salaris per afdeling berekend. Zoals we reeds vermeldden is het resultaat van een SELECT geordend op een manier, bepaald door het systeem en dit is daarom niet de ordening volgens afdelingsnummer. Hoe zou je ook nog ordening krijgen van elke afgedrukte groepslijn volgens afdeling?

```
SELECT Afd, MAX(Salaris), MIN(Salaris), AVG(Salaris)
FROM Werknemer
GROUP BY Afd
ORDER BY Afd
```

Opgave:

1. Geef per afdeling, het afdelingsnummer en het aantal Werknemers, gesorteerd volgens afdelingsnummer.
2. Idem, maar nu gesorteerd volgens aantal Werknemers.
3. Tel per afdeling het aantal mannen en vrouwen en sorteer volgens opklimmende afdeling en afdalend geslacht.

```
SELECT Afd, Gesl, COUNT(*)
FROM Werknemer
GROUP BY Afd, Gesl
ORDER BY Afd, Gesl DESC
```

Het is ook mogelijk een WHERE-clausule, samen met een GROUP BY-clausule te gebruiken. In dergelijk geval werkt de WHERE-clausule als een zeef die alle rijen, die niet aan de zoekvoorwaarde voldoen, doet verdwijnen. Daarna worden de groepen gevormd en worden de ingebouwde functies losgelaten op de overblijvende rijen van elke groep.

Voorbeeld:

```
SELECT Afd, MAX(Salaris), MIN(Salaris), AVG(Salaris)
FROM Werknemer
WHERE Geslacht = 'M'
GROUP BY Afd
```

Hier wordt dus het maximum, het minimum en het gemiddeld loon berekend voor alle mannelijke bedienden en dit per afdeling.

Indien we een WHERE-clausule samen met een GROUP BY-clausule gebruiken, beperken we in het algemeen het aantal rijen van de verschillende groepen. Wensen we echter het aantal groepen zelf te beperken, dan moeten we een conditie vermelden bij de selectie van de groepen. Dit wordt bekomen door aan GROUP BY een HAVING-conditie toe te voegen.

Dus:

Met de HAVING-clausule SELECTEERT of VERWERPT men GROEPEN.
Met de WHERE-clausule SELECTEERT of VERWERPT men individuele rijen.

Voorbeeld:

Geef een lijst die per afdeling het afdelingsnummer, het maximum, het minimum en het gemiddeld salaris bevat van de mannelijke personeelsleden.
Een afdeling komt enkel voor afdruk in aanmerking, indien ze meer dan 2 mannelijke bedienden bevat en indien het maximum mannelijk salaris meer dan 2 maal het minimum mannelijk salaris overtreft voor die afdeling.

```
SELECT Afd, MAX(Salaris), MIN(Salaris), AVG(Salaris)
FROM Werknemer
```

```
WHERE Geslacht='M'
GROUP BY Afd
HAVING COUNT (*) > 2 AND MAX(SALARIS) > 2 * MIN(SALARIS)
```

Opgave:

1. Geef een overzicht van de afdelingen die tenminste 2 Werknemers hebben die meer dan 50000 verdienen.
2. Geef een overzicht van de afdelingen die minstens 2 Werknemers hebben die pas vanaf 80 in dienst zijn.
3. Geef een overzicht van de afdelingen die meer dan 1 Werknemer hebben met opleidingsniveau 20.
4. Bedenk zelf een analoge vraag voor de leeftijd.

HOOFDSTUK 8

Werken met meerdere tabellen

Voor het oplossen van bepaalde vragen moeten soms verbanden tussen tabellen gelegd worden. Een manier waarop dit kan is het maken van een join tussen tabellen. Een andere manier is het gebruik van subqueries (zie 2^{de}jaar)

8.1 JOIN van verschillende tabellen

8.1.1 INNER JOIN van verschillende tabellen

Soms zijn de gegevens, vereist in een query, afkomstig uit verschillende tabellen. De verbanden tussen 2 tabellen worden meestal gelegd op basis van de relatie sleutel/vreemde sleutel.

Een join kan op 2 manieren gespecifieerd worden.

1. De manier voorgeschreven in de SQL-92 standaard :

   In de FROM-clausule worden de vereiste tabellen opgegeven, gescheiden door het keyword JOIN. (Je mag ook INNER JOIN gebruiken.)

   De ON-clause specifieert dan een bepaalde relatie tussen de rijen die moeten samengevoegd worden.

   Voorbeeld: De tabel Werknemer bevat geen afdelingsnamen. Als we een lijst willen maken die alle namen, afdelingsnummers en afdelingsnamen bevat, zullen we de 2 tabellen Werknemer en Afdeling moeten samenvoegen, mits de afdelingsnummers overeenstemmen. De (JOIN) relatie tussen beide tabellen wordt bijgevolg gelegd m.b.v. het afdelingsnummer.

   ```
   SELECT Vnaam, Fnaam, Afd, AfdNaam
   FROM Werknemer JOIN Afdeling ON Werknemer.afd = Afdeling.Afdnr
   ```

2. De "Old style join"

Een aantal DBMS zoals Oracle kennen enkel de Old style JOIN. Ook een groot aantal applicaties en voorbeelden gebruiken de old style join.

In de FROM-clausule worden enkel de vereiste tabellen opgegeven, gescheiden door komma's.

In de WHERE-clausule specificeren we dan een bepaalde voorwaarde tussen de rijen die moeten samengevoegd worden (JOIN)

Het voorbeeld wordt dan:

```
SELECT Vnaam, Fnaam, Afd, AfdNaam
FROM Werknemer, Afdeling
WHERE Werknemer.afd = Afdeling.Afdnr
```

Opmerking:

- Zonder WHERE krijgen we alle mogelijke combinaties van rijen uit de 1^{ste} tabel met rijen uit de 2^{de} tabel. (Dit is het voordeel van de ANSI-syntax. Als je de ON-clause vergeet, dan krijg je een syntax fout.)
- Conceptueel kunnen we ons voorstellen dat MySQL elke combinatie van rijen vormt met de opgegeven tabellen en daarna elke combinatie test op de conditie, voorkomend in de WHERE.

Algemene opmerkingen:

- Beide syntaxen worden door MySQL ondersteund.
- De prefix Werknemer en Afdeling zijn hier eigenlijk niet nodig aangezien de kolomnamen zelf reeds verschillend zijn. De oplossing wordt dan:

  ```
  SELECT Vnaam, Fnaam, Afd, AfdNaam
  FROM Werknemer, Afdeling
  WHERE Afd = Afdnr
  ```

  Hadden de kolommen waarover de join gebeurt in de 2 tabellen dezelfde naam, dan moet men kwalificeren, d.w.z. de tabelnamen als prefix toevoegen.
  Voorbeeld: Stel dat in de tabel Afdeling de kolom AfdNR ook de naam Afd had gedragen, dan zou vorige query luiden:

  ```
  SELECT Vnaam, Fnaam, Werknemer.Afd, AfdNaam
  FROM Werknemer, Afdeling
  WHERE Werknemer.Afd = Afdeling.Afd
  ```

- Je kan ook verkorte notaties voor tabellen gebruiken.

  Voorbeeld:

```
SELECT Vnaam, Fnaam, W.Afd, AfdNaam
FROM Werknemer W, Afdeling A
WHERE W.Afd = A.Afd
```

8.1.2 JOIN van een tabel met ZICHZELF

Soms is het vereist dat men een tabel met zichzelf moet samenvoegen. Om dit te begrijpen moet men zich 2 "virtuele" tabellen voorstellen, gebaseerd op dezelfde basistabel. In de FROM-clausule wordt de tabelnaam dan ook 2 maal opgegeven, om aan te duiden dat de join, bestaat uit combinaties van rijen genomen uit dezelfde tabel.
Er zou dus b.v.b. komen: FROM Werknemer, Werknemer
Daar nu de tabelnamen uniek moeten zijn, geeft men een uniek label aan beide, b.v.b. als volgt: FROM Werknemer X, Werknemer Y.

Kolommen geselecteerd uit de eerste tabel Werknemer worden gekwalificeerd met X, b.v.b. X.NR, X.Afd, enz.

Kolommen geselecteerd uit de tweede tabel Werknemer worden gekwalificeerd met Y, b.v.b. Y.TEL, enz.

Opgave: Selecteer de Werknemers paarsgewijze als volgt: ze werken in dezelfde afdeling, maar hun jobcode verschilt minstens 5.

Oplossing:

```
SELECT X.FNaam, X.Code, Y.FNaam, Y.Code
FROM Werknemer X JOIN Werknemer Y ON X.Afd = Y.Afd
WHERE X.Code >= Y.Code + 5
```

8.1.3 Outer JOIN

Inner Joins geven enkel die rijen terug die voldoen aan de ON conditie. Dit betekent dat als een rij in de eerste tabel niet overeenstemt met een rij uit de tweede tabel de rij niet zal geretourneerd worden. In bepaalde gevallen echter wensen we toch alle rijen van de ene tabel te zien eventueel gelinkt aan de gegevens van de andere tabel. Hiervoor gebruiken we een outer join.

Er zijn 3 types van outer join

- LEFT (OUTER) JOIN : retourneert alle rijen van de eerst genoemde tabel in de FROM clause.
- RIGHT (OUTER) JOIN : retourneert alle rijen van de tweede tabel in de FROM clause.

- FULL (OUTER) JOIN : retourneert ook rijen uit de eerste en tweede tabel die geen corresponderende entry hebben in andere tabel. Met andere woorden: de resultset bevat steeds alle rijen van de eerste tabel en alle rijen van de tweede tabel.

Voorbeeld :

We wensen per afdeling te weten hoeveel werknemers tot de afdeling behoren, ook voor de afdeling zonder werknemers. Afdeling D01 moet dus ook in het resultaat voorkomen met als aantal werknemers de waarde 0.

```
SELECT afdnr, afdNaam, count(FNaam)
FROM afdeling left outer join werknemer ON afd = afdnr
GROUP BY afdnr, afdNaam
ORDER BY afdnr
```

Opmerking : In de select lijst wordt count(Fnaam) gebruikt en niet count(*). Waarom?

8.1.4 CROSS JOIN

Een CROSS JOIN is vergelijkbaar met het cartesisch product. Het aantal rijen in de resultaattabel is gelijk aan het aantal rijen in de eerste tabel maal het aantal rijen in de tweede tabel.

```
SELECT Vnaam, Fnaam, afdnr, afdNaam
FROM Werknemer CROSS JOIN Afdeling ON afd = afdnr
```

Deze join wordt in de praktijk zelden gebruikt, maar kan handig zijn om testdata te genereren.

8.2 UNION

UNION combineert 2 of meerdere tabellen tot 1 tabel.

Voorbeeld :

Stel dat de tabel Werknemer opgesplitst wordt in 2 tabellen, een tabel WerknemersInDienst en een tabel WerknemersUitDienst. De gegevens uit beide tabellen kunnen worden samengevoegd a.d.h.v. een UNION.

```
    SELECT Vnaam, Fnaam
    FROM WerknemerInDienst
UNION
    SELECT Vnaam, Fnaam
    FROM WerknemerUitDienst
```

Om de gegevens uit twee of meerdere tabellen via een union te kunnen samenbrengen moeten wel een aantal regels gerespecteerd worden:

- De resultaten van de SELECT opdrachten moeten evenveel kolommen bevatten.
- Overeenkomstige kolommen uit de SELECTs moeten van hetzelfde data type zijn en NOT NULL toelaten of niet.
- Kolommen komen voor in dezelfde volgorde.
- De kolomnamen/titels van de UNION zijn deze van de eerste SELECT.
- Het resultaat bevat echter steeds alleen unieke rijen.
- Aan het einde van de UNION kan je een ORDER BY toevoegen. In deze clausule mag geen kolomnaam of uitdrukking voorkomen indien kolomnamen van beide selects verschillend zijn. Gebruik in dat geval kolomnummers.

HOOFDSTUK 9

Wijzigen van de inhoud van tabellen

Via SQL kunnen gegevens worden gewijzigd door nieuwe rijen in te voegen, door rijen te verwijderen of door waarden van kolommen in bestaande rijen te veranderen.

9.1 INSERT

9.1.1 Eén rij

Voorbeeld:

Veronderstel een nieuwe afdeling met nummer D41, naam = Systeem Test met Theo Spencer als manager. Alles is dus volledig om de tabel Afdeling aan te vullen.

Oplossing:

```
INSERT INTO Afdeling
VALUES ('D41','Systeem Test',100)
```

Als alles volledig is, en bovendien in de juiste volgorde opgegeven is, hoef je geen kolomnamen op te geven (tabelnaam volstaat).
Indien niet alle velden bekend zijn op het INSERT-tijdstip, kan je op 2 manieren te werk gaan:

- Ofwel vermeld je in de lijst van kolomnamen, enkel de kolomnamen, die een waarde zullen krijgen. De niet vermelde kolomnamen krijgen dan automatisch NULL-waarden toegewezen.
- Je vermeldt geen lijst van kolomnamen en gebruikt het NULL-sleutelwoord voor de onbekende velden.

Nr	Vnaam	Init	Fnaam	Afd	Tel	InDienst	Gesl	Salaris
(1)	(2)	(3)	(4)	(5)				

Tabel 9.1: Tabel AFDD41

Voorbeeld:

Twee nieuwe werknemers, waarvan enkel bekend is: Werknemersnummer, naam, afdeling, datum indiensttreding en geslacht.

1^{ste} manier:

```
INSERT INTO Werknemer(Nr, VNaam, Init, FNaam, Afd, Indienst, Gesl)
VALUES('410','Hugo','','Jansen','D41',820308,'M')
```

2^{de} manier:

```
INSERT INTO Werknemer
VALUES ('420','Caroline','H','Goderis','D41',NULL,820312,NULL,NULL,'V', NULL,NULL)
```

9.1.2 Meerdere rijen

We gaan hier eerst even in op de creatie van tabellen. Creatie van tabellen gebeurt d.m.v. de operator CREATE (zie in detail in volgend hoofdstuk).

Stel dat we een nieuwe tabel gecreëerd hebben, speciaal voor de nieuwe afdeling Systeem Test met naam AFDD41 (zie tabel 9.1).

Dit gebeurt door middel van:

```
CREATE TABLE AFDD41()
        NR CHAR(3) NOT NULL,                (1)
        VNaam VARCHAR(12) NOT NULL,         (2)
        Init CHAR(1) NOT NULL,              (3)
        FNaam VARCHAR(15) NOT NULL,         (4)
        Afd CHAR(3) NOT NULL,               (5)
        Tel CHAR(4),
        InDienst INT,
        Gesl CHAR(1),
        Salaris DECIMAL(8,2)
)
```

De eerste 5 kolommen kunnen geen NULL bevatten. Bij INSERT moet je dus steeds waarden opgeven voor deze kolommen.

Bij de laatste 4 kolommen zullen NULL-waarden ingevuld worden, indien je hiervoor geen data invult.

Opgave:

Vul AFDD41 nu op met de gegevens van de 2 bedienden uit 9.1.1 .
De 2 nieuwe bedienden hebben een nummer van de gedaante 4_ in tabel Werknemer na insert uit 9.1.1 hierboven. Bovendien weten we ook dat in tabel Afdeling de manager zit van D41. Via MANNR uit de rij met als AfdNR 'D41' uit de tabel Afdeling leggen we de relatie met tabel Werknemer. Zo bekomen we de 3^{de} werknemer ter invulling van AFDD41.

Oplossing:

```
INSERT INTO AFDD41(Nr, VNaam, Init, FNaam, Afd, Indienst, Gesl)
SELECT Nr, VNaam, Init, FNaam, Afd, Indienst, Gesl
FROM Werknemer
WHERE NR LIKE '4_'
      OR Nr IN (SELECT ManNr FROM Afdeling
            WHERE AfdNr = 'D41')
```

Merk op dat hier in de SELECT een SUBSELECT voorkomt.(zie 2^{de} jaar)

9.2 UPDATE

Doel: De waarden van 1 of meer kolommen uit één of meer rijen van een tabel wijzigen. De rij(en) die moet(en) gewijzigd worden, worden in de WHERE-clausule geselecteerd.

Voorbeeld:

Zet in Werknemer de jobcode op 52 en het salaris op 450 van de bedienden met nummer 410 en 420. Zet de datum van indiensttreding op NULL.

```
UPDATE Werknemer
SET Code = 52,  Salaris =450, InDienst = NULL
WHERE Nr = '410' OR Nr = '420'
```

Indien geen WHERE vermeld wordt, zullen alle rijen in de opgegeven tabel gewijzigd worden!!!

Voorbeeld:

Geef alle werknemers 7,5% opslag.

```
UPDATE Werknemer
SET Salaris= 1.075 * Salaris
```

9.3 DELETE

Stel dat de 'Systeem Test' afdeling opgedoekt wordt. Schrap de 2 werknemers uit Werknemer. Schrap de overeenkomstige afdelingsregel in de tabel Afdeling en schrap de volledige tabel AFDD41.

```
DELETE
FROM Werknemer
WHERE NR = '410' OR NR = '420'
```

```
DELETE
FROM Afdeling
WHERE AfdNr = 'D41'
```

Wat denk je van volgend alternatief:

```
DELETE
FROM Afdeling
WHERE ManNr IN (SELECT Nr FROM AFDD41)
```

Dit ter illustratie van het feit dat een DELETE ook een subselect kan bevatten.

Tenslotte moeten we alle rijen uit de tabel AFDD41 schrappen:

```
DELETE
FROM AFDD41
```

Merk op dat bij dit laatste SQL-statement, wel alle rijen geschrapt worden, maar dat de tabel AFDD41 toch nog als lege tabel blijft bestaan.

HOOFDSTUK 10

Aanmaken, wijzigen en verwijderen van tabellen

10.1 Creatie van een tabel

CREATE TABLE heeft een logische regelstructuur gescheiden door komma's. Eén logische regel bevat een volledige kolomdefinitie, of een constraint zoals de definitie van een primaire sleutel of vreemde sleutel.

Als je een tabel definieert moet je minimum het volgende specifiëren:

- Tabelnaam
- Kolomnamen
- Datatypes van kolommen
- NULL of NOT NULL

Voorbeeld:

```
CREATE TABLE AFDD41(
        NR CHAR(3) NOT NULL,
        VNaam VARCHAR(12) NOT NULL,
        Init CHAR(1) NOT NULL,
        FNaam VARCHAR(15) NOT NULL,
        Afd CHAR(3) NOT NULL,
        Tel CHAR(4) NULL,
        InDienst INT NULL,
        Gesl CHAR(1) NULL,
        Salaris DECIMAL(8,2) NULL
)
```

10.2 Wijzigen van de tabelstructuur

- *Toevoegen van een kolom*

  Je kan enkel een kolom toevoegen aan een niet lege tabel indien deze kolom NULL values toelaat of een default waarde bevat. Immers bij uitvoering van een ALTER op een bestaande tabel zullen alle bestaande rijen in de nieuwe kolom(men) NULL krijgen, wat zeker strijdig zou zijn met de NOT NULL uit de ALTER.

  Algemene vorm:

  ALTER TABLE tabelnaam
  ADD (kolomnaam type [, kolomnaam type])

  Voorbeeld :

  Voeg een kolom Adres toe aan de tabel AFDD41
  ALTER TABLE AFDD41
  ADD Adres varchar(40) NULL

- *Wijzigen van een kolom*

  ALTER TABLE AFDD41
  ALTER Adres varchar(50) NULL

- *Verwijderen van een kolom*

  Alle indexen en constraints op een kolom moeten eerst verwijderd zijn alvorens je de kolom kan verwijderen.
  ALTER TABLE AFDD41
  DROP COLUMN Adres

10.3 DROP : Verwijderen van een volledige tabel

Alles verdwijnt: niet alleen de tabeldefinitie, maar ook de indexen en de rechten gebouwd op deze tabel. (Dit is dus niet hetzelfde als een DELETE van alle rijen van een tabel.)
Je kan enkel een tabel verwijderen indien er geen afhankelijkheden (relaties) meer zijn met andere tabellen.

DROP TABLE ADFD41

10.4 AutoNumber velden (AUTO_INCREMENT)

Indien je de primaire sleutel automatisch door het systeem wenst te laten genereren, dan geef je het keyword AUTO_INCREMENT mee. Hierdoor worden de records genummerd vanaf 1 met een stapgrootte = 1.
Een AUTO_INCREMENT kolom laat geen NULL waarden toe.

Voorbeeld :

```
CREATE TABLE users(
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(40),
    password VARCHAR(255),
    email VARCHAR(255)
    );
```

Wens je de nummering te starten vanaf een andere waarde, dan geef je de startwaarde mee in het CREATE statement of gebruik je hiervoor een ALTER statement.

Voorbeeld:

```
ALTER TABLE users AUTO_INCREMENT = 100;
```

10.5 Definitie van constraints

Een constraint is in feite een beperking die je oplegt, en waarop gecontroleerd wordt bij het toevoegen, aanpassen en verwijderen van rijen.
Een tabeldefinitie bevat naast de definitie van kolommen ook de definitie van constraints. Allereerst heb je het NOT NULL constraint. Deze constraint wordt altijd gespecifieerd als onderdeel van de kolomdefinitie. Dit is een inline constraint. Andere constraints, kunnen zowel als inline of als out of line constraints, d.i. via een aparte logische regel, gedefinieerd worden.

De definitie van een out of line constraint:

```
CREATE TABLE of ALTER TABLE
CONSTRAINT constraint_naam type uitdrukking
```

De definitie van een inline constraint:

```
CREATE TABLE of ALTER TABLE
Veld_naam veld_type constraint_type constraint_uitdrukking
```

Mogelijke constraint types:

- NULL, NOT NULL
- PRIMARY KEY : definitie van de primaire sleutel.
- FOREIGN KEY : opleggen van referentiële integriteit
  Voorbeeld:
  CONSTRAINT Postc_fk FOREIGN KEY(Postc) REFERENCES Postcode(pc))
- CHECK : specificatie van toegelaten waarden in een kolom bij INSERT en UPDATE.

- UNIQUE : specifieert of de waarden binnen 1 of meerdere kolommen al dan niet uniek moeten zijn. B.v.b Rijksregisternummer.
- DEFAULT : specifieert de default waarde voor een kolom als bij INSERT geen waarde wordt opgegeven.

Voorbeeld:

```
CREATE TABLE AFDD41(
        Nr INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
        VNaam VARCHAR(12) NOT NULL,
        Init CHAR(1) NOT NULL ,
        FNaam VARCHAR(15) NOT NULL,
        Afd CHAR(3) NOT NULL,
        Tel CHAR(4) NULL,
        Postc CHAR(4) NULL,
        InDienst INT NULL,
        Gesl CHAR(1) DEFAULT 'M' CHECK(Gesl IN ('M','V')) NULL,
        Salaris DECIMAL(8,2) NULL
        CONSTRAINT Postc_fk FOREIGN KEY(Postc) REFERENCES Postcode(pc)
)
```

HOOFDSTUK 11

Sleutels

11.1 Primaire Sleutel en Entiteitsintegriteit

Van een tabel moet aangeduid worden welke kolom of combinatie van kolommen de *primaire sleutel* is.

De specificatie PRIMARY KEY geeft aan dat de geassocieerde kolom of combinatie van kolommen geen dubbele waarden mag bevatten. Hiermee is de *uniciteit* van de primaire sleutel verzekerd. Nogal wat systemen in de praktijk waarborgen nochtans deze voorwaarde enkel wanneer een UNIQUE INDEX is gedefinieerd over de (combinatie van) kolom(men) die als primaire sleutel optreedt.

De *entiteitsintegriteitregel* bepaalt dat de primaire sleutel geen NULL-waarden mag hebben. Een kolom of een combinatie van kolommen die als PRIMARY KEY gedefinieerd is, moet dus ook een NOT NULL-declaratie hebben.

Voorbeeld:

PRODUKT (PRODNR, PRODNAAM, HOEV_IN_VOORR)

```
CREATE TABLE PRODUKT(
        PRODNR SMALLINT NOT NULL PRIMARY KEY,
        PRODNAAM CHAR(16),
        HOEV_IN_VOORR SMALLINT
)
```

HOOFDSTUK **12**

View

Het begrip VIEW in SQL maakt het mogelijk, het resultaat van een SELECT als een echte tabel te beschouwen.

Doel: Het levert een middel om lange en complexe queries meer handelbaar te maken.

Voorbeeld:

Stel dat we in toepassingen steeds moeten werken met mannelijke personeelsleden uit afdeling D11 met jobcode 55 en opleidingsniveau 16.
In al deze toepassingen zal de WHERE er als volgt uitzien:

```
SELECT
FROM Werknemer
WHERE ... (overige condities)
      AND Afd = 'D114'
      AND CODE = 55
      AND NIV = 16
      AND GESL = 'M'
```

Een betere oplossing is: creëer een view van de tabel Werknemer, waarin alleen de records zichtbaar zijn die we nodig hebben.

Algemene vormen:

```
CREATE VIEW view-naam [(kolomnaam-lijst)] AS
   SELECT-statement


DROP VIEW view-naam


ALTER VIEW view-naam [(kolomnaam-lijst)] AS
```

SELECT-statement

Dus in ons voorbeeld wordt dit:

```
CREATE VIEW DEELTAB AS
    SELECT *
    FROM Werknemer
    WHERE Afd = 'D114'
        AND CODE = 55
        AND NIV = 16
        AND GESL = 'M'
```

De queries worden dan als volgt verder uitgevoerd op DEELTAB:

```
SELECT ...
FROM DEELTAB
WHERE ...
```

De geselecteerde tuples zullen dan zeker aan alle hierboven in vet aangeduide voorwaarden voldoen.

Door een VIEW kan men een combinatie van verschillende tabellen (join) beschouwen als één grote tabel.
Het vormt voor de DBA een instrument tot beveiliging van de DB.

BIJLAGE **A**

Appendix A

A.1 Databank Werknemer

Nr	Vnaam	Init	Fnaam	Afd	Tel	InDienst	Code	Niv	Gesl	GebDat	Salaris
010	Christine	I	Haas	A00	3978	650101	66	18	V	330814	1308
020	Michel	L	Theunis	B01	3476	731001	61	18	M	480202	1023
030	Sally	A	Kramer	C01	4738	750405	60	20	V	410511	948
050	Johan	B	Geysen	E01	6789	490817	58	16	M	250915	996
060	Irving	F	Steur	D11	6423	730914	55	16	M	450707	799
070	Eva	D	Pulanski	D21	7831	800930	56	16	V	530526	897
090	Evelien	W	Hendriks	E11	5498	700815	55	16	V	410515	737
100	Theo	Q	Spencer	E21	0972	800619	54	14	M	561218	648
110	Vincent	G	Leman	A00	2167	631205	58	19	M	291105	1153
120	Sean		Connors	A00	3490	580516	58	14	M	421018	725
130	Danielle	M	Scheire	C01	4578	710728	55	16	V	250915	590
140	Hilde	A	Nagels	C01	1793	761215	56	18	V	460119	705
150	Bruno		Adams	D11	4510	720212	55	16	M	470517	627
160	Els	R	Placke	D11	3782	771011	54	17	V	550412	552
170	Mats	J	Sierens	D11	2890	780915	54	16	M	510105	612
180	Marleen	S	Schouters	D11	1682	730707	53	17	V	490221	529
190	Jan	E	Wauters	D11	2986	740726	53	16	M	520625	507
200	David		De Bruyn	D11	4501	660303	55	16	M	410529	688
210	Willem	T	Jansens	D11	0942	790411	25	17	M	530223	453
220	Jennifer	K	Luyckx	D11	0672	680829	55	18	V	480319	740

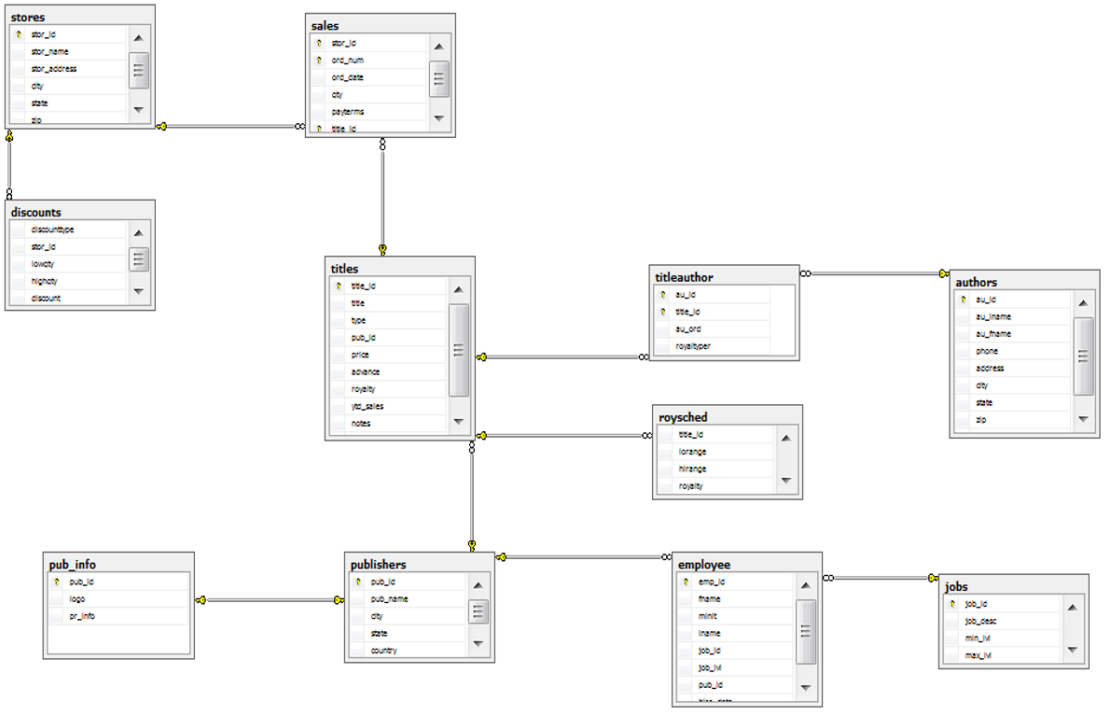
Tabel A.1: Tabel Werknemer

Opmerking: ManNr is het werknemersnummer van de manager van de afdeling.

AfdNr	AfdNaam	ManNr
A00	Computer Service	010
B01	Planning	020
C01	Informatie Centr.	030
D01	Ontwikkelingscentr.	050
E01	Support Services	060
D11	Administratie	070
D21	Software support	100
E21	Tools	090

Tabel A.2: Tabel Afdeling

A.2 Template van de PUBS databank



Figuur A.1: ERD van de PUBS databank

Lijst van figuren

Lijst van tabellen