

Het ontwerpen van een databaseschema

Maarten Fokkinga

Versie van 9 oktober 2008, 10:14

In deze verhandeling geef ik een notatie en methode om een databaseschema te ontwerpen. Ik onderscheid in grote lijnen de volgende stappen:

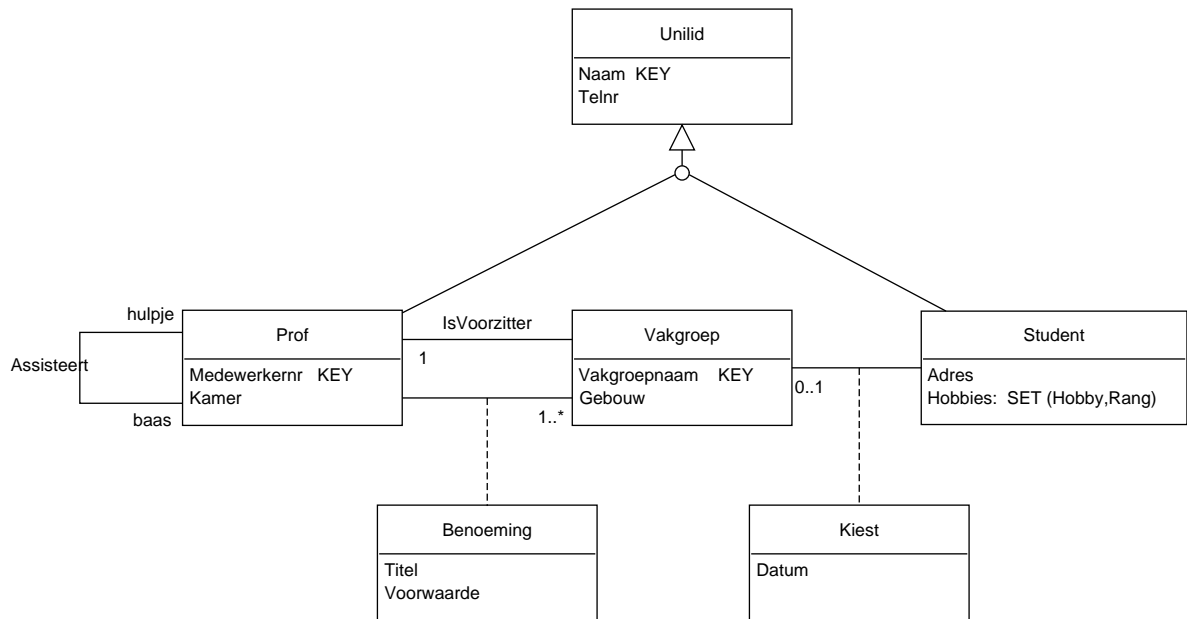
- Beschrijf de werkelijkheid met een ER-diagram. Gebruik de UML notatie en zo nodig gestructureerde attribuutwaarden, zoals **set**(..., **set**(...), ...) (dus met geneste structuur).
- Vertaal het ER-diagram naar een databaseschema door, zonder uitzondering, voor *ieder* entiteitstype en *iedere* relatie een tabel te nemen.
 - Ten gevolge van gestructureerde attribuutwaarden is het schema niet in “eerste normaalvorm” (1NF).
 - Sommige multipliciteiten kunnen gerepresenteerd worden door key en foreign key constraints; sommige eigenschappen kunnen alleen maar als checks (binnen tabellen) of asserties (buiten tabellen) vertaald worden.
- Normaliseer het databaseschema naar eerste normaalvorm (alle attributen hebben niet-gestructureerde waarden).
- Optimaliseer het databaseschema door sommige tabellen samen te nemen.
- Normaliseer verder naar Boyce-Codd normaalvorm en 4e normaalvorm.

We leggen deze stappen en de aanbevolen notaties uit aan de hand van een voorbeeld.

Opmerking. Lewis en Kifer kennen in de ER-modellering weliswaar **set**-waardige attributen maar geen anders-gestructureerde waarden. Hun vertaling van een ER-diagram geeft een databaseschema in eerste normaalvorm, maar dat heeft dan wel ongewenste MVDs (en onwerkelijke sleutels) die later met normalisatie naar 4NF worden weggevoerd. Mijn vertaling geeft een schema dat niet in eerste normaalvorm staat, en op eenvoudige wijze in stap 2 direct tot eerste normaalvorm genormaliseerd wordt.

1 ER-modellering

Ons voorbeeld speelt zich af op de universiteit: er zijn professoren, studenten, vakgroepen, en relaties tussen deze. De precieze situatie beschrijven we met opzet niet: die moet ook duidelijk worden uit het ER-diagram dat we van de situatie geven (en dat bedoeld is om de situatie te beschrijven). Ons ER-diagram van de casus staat, in de UML-notatie, in Figuur 1. We zullen eerst het diagram in detail bespreken en daarna de semantiek (betekenis en bedoeld gebruik) van een ER-diagram definiëren.



Figuur 1: ER-diagram van de universiteitscasus

Uitleg van het ER-diagram (*class* diagram, in UML terminologie).

- Terminologie.
 - De blokken heten *entiteitstypen* (hier: *Unilid*, *Prof*, *Vakgroep*, *Student*, *Benoeming*, en *Kiest*).
 - De doorgetrokken lijnen tussen blokken heten *relaties* (in UML terminologie: *associaties*). De naamloze relaties krijgen de naam van het eraan-hangende entiteitstype: hier *Benoeming* en *Kiest*. (In dit diagram zijn er dus vier relaties: *IsVoorzitter*, *Benoeming*, *Kiest* en *Assisteert*.)
 - Het bolletje-met-pijl heet generalisatie/specialisatie-knoop.
- Over entiteitstype *Unilid*:
 - Een instantie van *Unilid* is een verzameling waarvan we ieder lid ‘unilid’ noemen. De interpretatie van deze dingen wordt gesuggereerd door de naamgeving: leden van een universitaire gemeenschap.
 - Er is gedeclareerd dat *Naam* en *Telnr* attributen zijn. Dat betekent: ieder unilid heeft een ‘naam’ en ‘telnr’. De interpretatie van deze dingen wordt gesuggereerd door de naamgeving: een persoonsnaam, respectievelijk telefoonnummer.
 - Er is gedeclareerd dat (*Naam*) een sleutel is. Dat betekent: altijd bestaat er bij gegeven naam hooguit één unilid met die naam. We zeggen ook wel dat *Naam* de uniliden *identificeert*. Hopelijk is dit *onder de gesuggereerde interpretatie* waar (en zo niet: dan hebben we een ‘fout’ ER-diagram voor onze toepassing).

- Net zo voor entiteitstypen *Prof*, *Vakgroep* en *Student*, en voor de bijzondere entiteitstypen *Benoeming* en *Kiest*. Merk op dat binnen de blokjes voor *Student*, *Benoeming* en *Kiest* geen sleutel is gedeclareerd; binnen *Prof* en *Vakgroep* is dat wel het geval. (Toch definieert *dit* diagram ook voor *Student*, *Benoeming* en *Kiest* een sleutel, zoals we verderop zullen zien.)
- Over de generalisatie/specialisatie-knoop:
 - *Prof* is een specialisatie van *Unilid*, en *Unilid* is een generalisatie van *Prof*. Dit betekent: altijd is een prof ook een unilid; een prof heeft álle eigenschappen van een unilid. Een gevolg hiervan is: *Naam* is ook een attribuut, sleutel zelfs, van *Prof*, en *Telnr* is ook een attribuut van *Prof*.
 - Net zo voor *Student*; dus *Naam* is ook een attribuut, sleutel zelfs, van *Student* en *Telnr* is ook een attribuut van *Student*.
 - Bij de generalisatie/specialisatie-knoop mag een toevoeging *covering* staan, of *disjunct* of beide. De aanduiding *covering* betekent: altijd is een unilid ook prof of student of beide. De aanduiding *disjunct* betekent: nooit is een unilid zowel prof als ook student. In ons diagram staan geen van deze twee aanduidingen erbij, zodat er volgens dit diagram soms unileden bestaan die geen student zijn en ook geen prof, en er soms unileden bestaan die zowel student zijn als ook prof.
 - In databasetheorie heet de generalisatie/specialisatie-relatie ook wel ‘isA’ relatie (van het Engels: een prof ‘is a’ unilid).
- Over *Prof*. Hierin is (*Medewerkernr*) als sleutel gedeclareerd, terwijl (*Naam*) ook al sleutel is van *Prof*. Dus *Prof* heeft twee, alternatieve, sleutels.
- Over *Student*. Hierin is *Hobbies* gedeclareerd als een gestructureerd attribuut: één waarde voor *Hobbies* bestaat uit een verzameling van paren van een *Hobby* met een *Rang*. Voor iedere student wordt de rangschikking van de hobby in de voorkeurlijst van de student aangegeven door de waarde van *Rang*.
 In dit diagram zijn alle andere attribuutwaarden niet-gestructureerd (althans, er is geen structuur aangegeven), maar met enige fantasie zijn andere gestructureerde attributen goed denkbaar: vervang *Telnr* door *Telnrs* : set(*Netnr*, *Abonneer*), of *Gebouw* door *Lokatie* : (*Gebouw*, *Vloer*, *Vleugel*), of de alreeds gestructureerde *Hobbies* door de dieper gestructureerde *Hobbyactiviteiten* : set(*Activiteit*, *Gereedschap*, set(*Datum*, *Rng*)), enzovoort.
- Tussen *Prof* en *Vakgroep* bestaat er een relatie *IsVoorzitter*. Een instantie van zo’n relatie bestaat uit paren van prof en vakgroep. (Let op: zo’n tweetal zit in de relatie of niet; zoiets als “meermalen” in een relatie zitten is onzin.) Dus een sleutel van *IsVoorzitter* bestaat uit het paar van een sleutel van *Prof* en een sleutel van *Vakgroep*.
- Tussen *Prof* en *Vakgroep* bestaat er nóg een relatie, *Benoeming*. Een instantie van zo’n relatie bestaat uit paren van prof en vakgroep. Dus een sleutel van dit entiteitstype bestaat uit het paar van een sleutel van *Prof* en een sleutel van *Vakgroep*.
 Met ieder tweetal van een prof en vakgroep in de relatie *Benoeming* zijn een titel en een voorwaarde geassocieerd; dit is gedeclareerd in het entiteitstype *Benoeming* die, door

de stippellijn, identiek is aan de relatie *Benoeming*. De betekenis van het diagram verandert niet wanneer we *Benoeming* óók nog, of alléén maar, bij de relatie-lijn zetten.

- Tussen *Vakgroep* en *Student* bestaat er een relatie, *Kiest*. Net als bij *Benoeming* tussen *Prof* en *Vakgroep*, is ook deze relatie identiek aan het entiteitstype *Kiest*, zodat de declaratie van het attribuut *Datum* in het diagram opgeschreven kan worden. Met een notatie die we hier niet behandelen kan desgewenst de ‘leesrichting’ worden aangegeven: een student kiest een vakgroep (en niet andersom).
- Tussen *Prof* en *Prof* bestaat er een relatie *Assisteert*. Een instantie van zo’n relatie bestaat uit paren van prof en prof. In iedere relatie kan de *rol* van de deelnemers met een *rolnaam* worden aangegeven. In dit geval is dat gebeurd, en zijn de rolnamen *baas* en *hulpje* en is de bedoelde leesrichting zó: een hulpje-prof is de assistent van een baas-prof.
- De multipliciteit ‘1’ (of te wel ‘1..1’) bij de relatie *IsVoorzitter* betekent dat er altijd voor iedere vakgroep precies 1 prof bestaat die de vakgroep voorziet. Hopelijk is dit *onder de gesuggereerde interpretatie* waar (en zo niet: dan hebben we een ‘fout’ ER-diagram voor onze toepassing).

De multipliciteit ‘0..1’ bij de relatie *Kiest* betekent dat er altijd voor iedere student 0 of 1 vakgroepen bestaan die de student kiest. Hopelijk is dit *onder de gesuggereerde interpretatie* waar (en zo niet: dan hebben we een ‘fout’ ER-diagram voor onze toepassing).

De multipliciteit ‘1..*’ bij de relatie *Benoeming* betekent dat er altijd voor iedere prof 1 of meer vakgroepen bestaan waarin de prof benoemd is. Hopelijk is dit *onder de gesuggereerde interpretatie* waar (en zo niet: dan hebben we een ‘fout’ ER-diagram voor onze toepassing).

In een multipliciteit wordt ‘*’ gebruikt als afkorting van ‘oneindig’. De multipliciteiten die veel voorkomen (en nuttig zijn) zijn deze: 0..* (“zonder beperkingen”), 0..1 (“hoogstens één” of te wel de relatie is “functioneel”), 1..* (“minstens één”, of te wel de relatie is “totaal”), en 1..1 (“precies één”, of te wel de relatie is een “totale functie”). Sommige auteurs, waaronder ik zelf, beschouwen 0..* (“zonder beperkingen”) als default multipliciteit terwijl anderen een niet-geschreven multipliciteit als “niet-gespecificeerd” beschouwen.

De semantiek van een ER-diagram.

- Een *instantie van een entiteitstype* is een verzameling van *entiteiten*. Bijvoorbeeld, een instantie van *Prof* is een verzameling, en een lid daarvan is een entiteit en noemen we *Prof*-entiteit, of *Prof*-instantie of kortweg ‘prof’. Nog een voorbeeld: een instantie van *Benoeming* is een verzameling, en een lid daarvan noemen we *Benoeming*-instantie, of kortweg ‘benoeming’. Deze conventie hebben we hierboven en verderop consequent gebruikt.
- Een *instantie van een diagram* bestaat uit zódanige instanties voor de entiteitstypen en relaties, dat alle eigenschappen vervuld zijn: de multipliciteitseigenschappen (aangeduid door de getallen bij de relaties), de generalisatie/specialisatie-eigenschappen (aangeduid

door het rondje-met-open-pijltje) en de andere eigenschappen (als een *notitie* toegevoegd aan het ER-diagram).

In de uitleg van de betekenis van een ER-diagram zeggen we ‘altijd’ voor ‘in iedere instantie van het diagram’; we zeggen ‘nooit’ voor ‘in geen enkele instantie van het diagram’, en ‘misschien’ of ‘soms’ voor ‘in sommige instanties van het diagram’. Vaak laten we ‘altijd’ weg.

- We zeggen dat een ER-diagram ‘correct’ is wanneer van elke diagram-instantie de gesuggereerde interpretatie in voldoende mate overeenstemt met de werkelijkheid, dat wil zeggen, voldoende voor de bedoelde toepassingen. Correctheid is niet formeel te bewijzen (de werkelijkheid is niet te formaliseren); incorrectheid is wel aan te tonen, door het geven van een “tegenvoorbeeld” uit de werkelijkheid.

Ons ER-diagram zou incorrect genoemd kunnen worden omdat in een diagram-instantie een unilid zowel prof als ook student kan zijn, of omdat een prof z’n eigen hulpje kan zijn, of omdat een prof een vakgroep kan voorzitten waarin hij geen benoeming heeft. Alleen met de bedoelde toepassing voor ogen kan besloten worden of deze instanties werkelijk ongewenst zijn of niet. Desgewenst kunnen we deze instanties uitsluiten door extra eigenschappen te geven (dat wil zeggen, ergens als notitie aan het ER-diagram toe te voegen). Bijvoorbeeld: de al genoemde *disjoint* eigenschap, of de verderop behandelde cykel-eigenschappen.

- LET OP! Een heel belangrijk aspect van de correctheid van een ER-diagram is dat de entiteitstypen corresponderen met dingen uit de werkelijkheid *waarover* we informatie (onder andere ‘het bestaan’ van de entiteiten) willen bijhouden. Bijvoorbeeld, ons ER-diagram is geschikt om het bestaan van profs bij te houden, en van profs ook hun kamer. Ons ER-diagram is niet geschikt om het bestaan van hobby’s bij te houden; volgens het resulterende databaseschema zullen er *alleen* hobby’s opgeslagen kunnen worden die door een student beoefend worden! Hadden we wel het bestaan van hobby’s willen bijhouden, dan had *Hobby* niet een attribuut maar een entiteitstype moeten zijn en was het diagram complexer geweest: meer entiteitstypen en meer relaties.

2 Van ER-diagram naar DB-schema

Een *databaseschema* beschrijft de vorm en eigenschappen van *tabellen* die voor de opslag van waarden gebruikt gaan worden. In het hier gebruikte begrip van tabel mag geen betekenis gehecht worden aan de volgorde van rijen; de kolommen hebben een naam en ook aan hun volgorde mag geen betekenis toegekend worden. Vaak wordt een databaseschema grafisch weergegeven (en lijkt dan wel op een ER-diagram), waarbij iedere eigenschap “*dit* verwijst naar *dat*” met een pijl getekend wordt, en iedere eigenschap “*dit* is een sleutel” met een label *KEY* aangeduid wordt.

Terminologie. Een *instantie* van een databaseschema of tabelschema is een vulling met waarden. Een *databaseschema* beschrijft de vorm en eigenschappen van alle tabellen samen; de eigenschappen worden als eis gesteld aan *alle* instanties (= “altijd”). Een *tabelschema* beschrijft de vorm en eigenschappen van één tabel; iedere eigenschap is in feite een eigenschap van *alle* instanties van het schema (= “altijd”). Vaak laten we de aanduiding “altijd” weg

(zoals we hierboven overal gedaan hebben) en gebruiken we kortweg het woord *tabel*, in de verwachting dat uit de context duidelijk is of we *tabelschema* of *tabelinstantie* bedoelen.

Het schema – versie 1. We vertalen het ER-diagram naar een databaseschema door voor *ieder* entiteitstype en *iedere* relatie een tabel te nemen, en allereerst van de eigenschappen alleen de generalisatie mee te nemen; de multipliciteiten en expliciete eigenschappen worden alleen “overgeschreven” en worden later in de structuur verwerkt. In de vertaling zijn er op sommige plaatsen alternatieve mogelijkheden, en moeten er keuzen worden gemaakt. Sommige keuzen komen pas in Sectie 8 aan bod.

Het aldus verkregen databaseschema is ‘correct’ wanneer het ER-diagram ‘correct’ is. Het databaseschema is nog niet optimaal; een optimalisatiestap volgt later. De volledige vertaling staat *geformuleerd in woorden* in Figuur 2 en *getekend* in Figuur 3 (bladzijde 9).

| |
|---|
| <p><i>Unilid</i> (<i>Naam</i>, <i>Telnr</i>).</p> <p>(<i>Naam</i>) is sleutel.</p> <p><i>Prof</i> (<i>Naam</i>, <i>Medewerkernr</i>, <i>Kamer</i>).</p> <p>(<i>Naam</i>) is sleutel.</p> <p>(<i>Medewerkernr</i>) is sleutel.</p> <p><i>Naam</i> verwijst naar <i>Unilid</i>(<i>Naam</i>).</p> <p><i>Vakgroep</i> (<i>Vakgroepnaam</i>, <i>Gebouw</i>).</p> <p>(<i>Vakgroepnaam</i>) is sleutel.</p> <p><i>Student</i>(<i>Naam</i>, <i>Adres</i>, <i>Hobbies</i> : set(<i>Hobby</i>, <i>Rang</i>)).</p> <p>(<i>Naam</i>) is sleutel.</p> <p><i>Naam</i> verwijst naar <i>Unilid</i>(<i>Naam</i>).</p> <p><i>Benoeming</i>(<i>Naam</i>, <i>Vakgroepnaam</i>, <i>Titel</i>, <i>Voorwaarde</i>).</p> <p>(<i>Naam</i>, <i>Vakgroepnaam</i>) is sleutel.</p> <p><i>Naam</i> verwijst naar <i>Prof</i>(<i>Naam</i>).</p> <p><i>Vakgroepnaam</i> verwijst naar <i>Vakgroep</i>(<i>Vakgroepnaam</i>).</p> <p>Bij iedere <i>Prof</i>-rij (<i>n</i>, ...) is er minstens een <i>Vakgroep</i>-rij (<i>v</i>, ...) met (<i>n</i>, <i>v</i>, ...) in <i>Benoeming</i>.</p> <p><i>IsVoorzitter</i>(<i>Naam</i>, <i>Vakgroepnaam</i>).</p> <p>(<i>Naam</i>, <i>Vakgroepnaam</i>) is sleutel (wordt later verbeterd).</p> <p><i>Naam</i> verwijst naar <i>Prof</i>(<i>Naam</i>).</p> <p><i>Vakgroepnaam</i> verwijst naar <i>Vakgroep</i>(<i>Vakgroepnaam</i>).</p> <p>Bij iedere <i>Vakgroep</i>-rij (<i>v</i>, ...) is er precies een <i>Prof</i>-rij (<i>n</i>, ...) met (<i>n</i>, <i>v</i>) in <i>IsVoorzitter</i>.</p> <p><i>Kiest</i>(<i>Vakgroepnaam</i>, <i>Naam</i>, <i>Datum</i>).</p> <p>(<i>Vakgroepnaam</i>, <i>Naam</i>) is sleutel (wordt later verbeterd).</p> <p><i>Vakgroepnaam</i> verwijst naar <i>Vakgroep</i>(<i>Vakgroepnaam</i>).</p> <p><i>Naam</i> verwijst naar <i>Student</i>(<i>Naam</i>).</p> <p>Bij iedere <i>Student</i>-rij (<i>n</i>, ...) is er hooguit een <i>Vakgroep</i>-rij (<i>v</i>, ...) met (<i>v</i>, <i>n</i>, ...) in <i>Kiest</i>.</p> <p><i>Assisteert</i>(<i>Baas</i>, <i>Hulpje</i>).</p> <p>(<i>Baas</i>, <i>Hulpje</i>) is sleutel.</p> <p><i>Baas</i> verwijst naar <i>Prof</i>(<i>Medewerkernr</i>).</p> <p><i>Hulpje</i> verwijst naar <i>Prof</i>(<i>Medewerkernr</i>).</p> |
|---|

Figuur 2: Eerste databaseschema verkregen uit het ER-diagram van Figuur 1. Zie ook de toelichting in de tekst.

Hier is een toelichting op de tabellen en eisen van het databaseschema:

- Tabel *Unilid*:
 - Het beoogde gebruik van de tabel luidt als volgt:
Een unilid met naam n en telnr t wordt gerepresenteerd door een rij (n, t) in deze tabel, en er zijn geen andere rijen in de tabel.
 - De declaratie dat $(Naam)$ een sleutel is betekent: rijen met dezelfde $Naam$ -waarden zijn helemaal gelijk. Deze eis komt rechtstreeks uit het ER-diagram. Als die eis niet overeenstemt met de werkelijkheid, is er bij het opstellen van het ER-diagram al een fout gemaakt.
- Tabel *Prof*:
 - Het beoogde gebruik van de tabel luidt als volgt:
Een prof met naam n , telnr t , medewerkernr m en kamer k wordt gerepresenteerd door een rij (n, m, k) , en er zijn geen andere rijen in de tabel. Het telnr van een prof wordt in *Unilid* gerepresenteerd.
 - De declaratie dat $(Naam)$ een sleutel is betekent: rijen met dezelfde $Naam$ -waarden zijn helemaal gelijk. Deze eis komt rechtstreeks uit het ER-diagram. Als die eis niet overeenstemt met de werkelijkheid, is er bij het opstellen van het ER-diagram al een fout gemaakt.
 - De declaratie dat $(Medewerkernr)$ een sleutel is betekent: rijen met dezelfde $Medewerkernr$ -waarden zijn helemaal gelijk. Deze eis komt rechtstreeks uit het ER-diagram. Als die eis niet overeenstemt met de werkelijkheid, is er bij het opstellen van het ER-diagram al een fout gemaakt.
 - De declaratie dat $Naam$ verwijst naar $Unilid(Naam)$ betekent: voor iedere rij (n, m, k) in de tabel is er minstens één *Unilid*-rij (n', t') met $n=n'$. Deze eis is de vertaling van de generalisatie/specialisatie in het ER-diagram tezamen met de zojuist gemaakte representatiekeuze.
- Tabel *Vakgroep*: net zo.
- Tabel *Student*:
 - Het beoogde gebruik van de tabel luidt als volgt:
Een student met naam n , telnr t , adres a en hobbies $\{(h_1, r_1), \dots, (h_k, r_k)\}$ wordt gerepresenteerd door een rij $(n, a, \{(h_1, r_1), \dots, (h_k, r_k)\})$, en er zijn geen andere rijen in de tabel. Het telnr van student wordt in *Unilid* gerepresenteerd.
 - $(Naam)$ is een sleutel en verwijst naar $Unilid(Naam)$: net als bij *Prof*.
 - Het verschijnsel dat *Hobbies* een gestructureerde waarde heeft, wordt straks uit het databaseschema weggewerkt.
- Tabel *Benoeming*.
 - Het beoogde gebruik:
We kiezen ervoor om een prof met z'n naam te representeren (we hadden ook z'n medewerkernr kunnen nemen). Dus:

Een benoeming met titel t_0 en voorwaarde w_0 van een prof (n, t, m, k) in een vakgroep (v, g) wordt in de tabel gerepresenteerd door een rij (n, v, t_0, w_0) , en er zijn geen andere rijen in de tabel.

- De declaratie dat $(Naam, Vakgroepnaam)$ een sleutel is betekent: rijen met gelijke waarden voor $(Naam, Vakgroepnaam)$ zijn geheel gelijk.
- De declaratie dat $Naam$ verwijst naar $Prof(Naam)$ betekent: voor iedere rij (n, v, t, w) in de tabel is er minstens één $Prof$ -rij (n', m', k') met $n=n'$.

- Tabel *IsVoorzitter*.

- Het beoogde gebruik van de tabel luidt als volgt:
We kiezen ervoor om een prof met z'n naam te representeren (we hadden ook z'n medewerkernr kunnen nemen). Dus:

Het voorzitterschap van een prof (n, t, m, k) in een vakgroep (v, g) wordt in de tabel gerepresenteerd door een rij (n, v) , en er zijn geen andere rijen in de tabel.

- De eigenschap dat $(Naam, Vakgroepnaam)$ een sleutel is betekent: rijen met gelijke waarden voor $(Naam, Vakgroepnaam)$ zijn geheel gelijk.
- De verwijzingseisen spreken voor zich.
- Ook de vierde eis komt rechtstreeks uit het ER-diagram (de multipliciteit bij de relatie *IsVoorzitter*). Die zullen we verderop gebruiken om het databaseschema kwalitatief te verbeteren.

- Tabel *Kiest*.

- Analooq aan *Benoeming*.
- De vierde eis bij de tabel *Kiest* komt rechtstreeks uit het ER-diagram (de multipliciteit bij de relatie *Kiest*). Die zullen we verderop gebruiken om het databaseschema kwalitatief te verbeteren.

- Tabel *Assisteert*.

- Beoogd gebruik van de tabel:
We kiezen er voor om een prof door z'n naam te representeren. Dus:

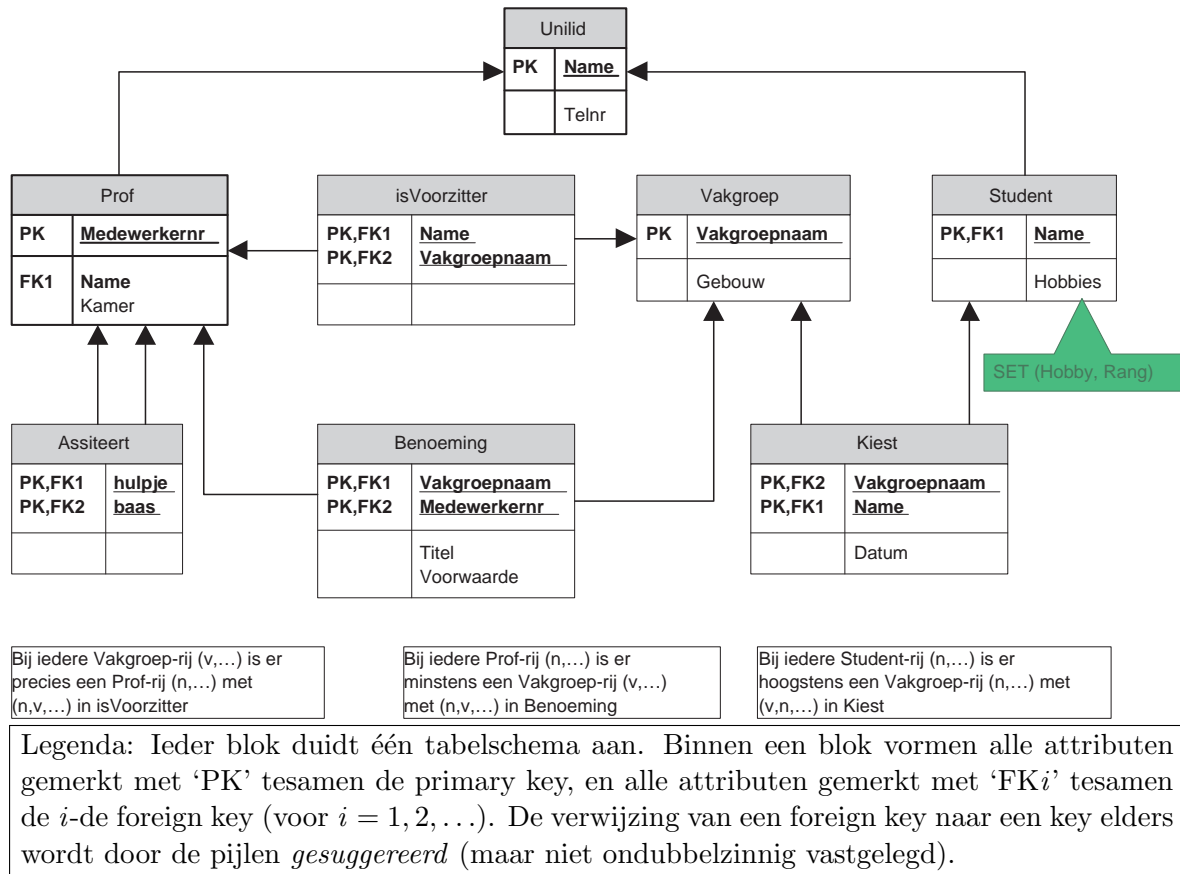
Het feit dat prof (n_0, t_0, m_0, k_0) als hulpje de assistent is van de baas (n_1, t_1, m_1, k_1) , wordt gerepresenteerd door de rij (n_0, n_1) , en er zijn geen andere rijen in de tabel.

- De verwijzingseisen spreken voor zich.

Voor de generalisatie/specialisatie hebben we hierboven een keuze gemaakt zonder duidelijk te zeggen dat er alternatieven waren. We zullen in Sectie 8 de alternatieven daarvoor behandelen.

De tekening in Figuur 3 van het zojuist besproken databaseschema lijkt erg op het ER-diagram van Figuur 1 maar heeft wel een paar belangrijke verschillen. Een klein verschil met het ER-diagram en met het besproken databaseschema is dat per tabel één sleutel is aangegeven als “primary key”. Een groot verschil met het ER-diagram is dat in het ER-diagram, bijvoorbeeld, *IsVoorzitter* een relatie is met deelnemende entiteiten *Prof* en *Vakgroep* (zij zijn

niet als attribuut vermeld in *IsVoorzitter*) terwijl in het databaseschema *IsVoorzitter* een tabel is net als alle andere en de keys van tabel *Prof* en *Vakgroep* wel als attribuut heeft (en zelfs als foreign key). Nog een groot verschil met het ER-diagram is dat in deze grafische weergave de lijnen (pijlen) geen relaties of associaties aanduiden, maar foreign key verwijzingen.



Figuur 3: Grafische weergave van het eerste databaseschema (zie Figuur 2) verkregen uit het ER-diagram van Figuur 1. De pijlen duiden niet zozeer “relaties” aan, maar foreign key verwijzingen.

3 Multipliciteiten

De drie multipliciteiten 0..1, 1..* en 1..1 (= 1) hebben het volgende effect op het databaseschema:

- Bij iedere *Student*-rij (n, \dots) is er hooguit een *Vakgroep*-rij (v, \dots) met (v, n, \dots) in *Kiest*.

Deze eigenschap is equivalent met: *Kiest*-rijen met gelijke waarden voor *Naam* zijn helemaal gelijk. En dit is weer equivalent met:

in *Kiest*: (*Naam*) een sleutel.

Er stond in versie 1 van het databaseschema dat in *Kiest* het paar (*Vakgroep*, *Naam*) een sleutel is, maar dat blijkt nu dus een supersleutel te zijn.

- Bij iedere *Prof*-rij (n, \dots) is er minstens een *Vakgroep*-rij (v, \dots) met (n, v, \dots) in *Benoeming*.

Deze eigenschap is equivalent met: Bij iedere *Prof*-rij (n, \dots) is er minstens een (n', v, \dots) in *Benoeming* met $n' = n$. Dit is equivalent met: *Prof*(*Naam*) verwijst naar *Benoeming*(*Naam*). Omdat in *Benoeming* attribuut *Naam* geen sleutel is, is de eigenschap geen foreign key constraint. De manier om de eigenschap uit te drukken is het definiëren van:

- * binnen tabel *Prof*: `check (Naam in (select B.Naam from Benoeming B))`
- * of, buiten alle tabellen: een geschikte *assertion*.

- Bij iedere *Vakgroep*-rij (v, \dots) is er precies een *Prof*-rij (n, \dots) met (n, v) in *IsVoorzitter*.

“Precies een” is de conjunctie van “hoogstens een” en “minstens een”. Dus enerzijds is in *IsVoorzitter* attribuut *Vakgroepnaam* een sleutel. (Er stond in versie 1 van het databaseschema dat in *IsVoorzitter* het paar $(Naam, Vakgroep)$ een sleutel is, maar dat blijkt nu dus een supersleutel te zijn.) Anderzijds is in *Vakgroep* het attribuut *Vakgroepnaam* een verwijzing naar *IsVoorzitter*(*Vakgroepnaam*). Dankzij het feit dat $(Vakgroepnaam)$ een sleutel is in *IsVoorzitter*, is de verwijzingeigenschap een foreign key constraint:

in *Prof*: `foreign key Naam references Benoeming(Naam)`.

4 Normalisatie tot 1NF

SQL, de taal waarmee we uiteindelijk de database willen implementeren, kan niet omgaan met gestructureerde waarden voor attributen, zoals `set(Hobby, Rang)`-waarden voor attribuut *Hobbies*. Die moeten we dus wegwerken; in ruil daarvoor verschijnen er nieuwe tabellen. Het resulterende schema heet “in eerste normaalvorm” te staan, 1NF. Deze normalisatie tot 1NF gaat als volgt.

Beschouw, in het schema van Figuur 2, een tabel met gestructureerde attribuutwaarden:

Student(*Naam*, *Adres*, *Hobbies* : `set`(*Hobby*, *Rang*))

(*Naam*) is sleutel.

Naam verwijst naar *Unilid*(*Naam*).

Vervang die ene tabel door deze twee:

*Student*₀(*Naam*, *Adres*)

(*Naam*) is sleutel.

Naam verwijst naar *Unilid*(*Naam*).

*Student*₁(*Naam*, *Hobby*, *Rang*)

(*Naam*, *Hobby*) is sleutel.

Naam verwijst naar *Unilid*(*Naam*).

(*Rang* is niet opgenomen in de sleutel, omdat volgens de eerder gegeven toelichting op bladzijde 3 van iedere student en hobby de rang vast ligt.)

Het bedoelde gebruik van *Student*₀ en *Student*₁ ligt nu voor de hand:

Een student met naam n , telnr t , adres a en hobbies $\{(h_1, r_1), \dots, (h_k, r_k)\}$ wordt gerepresenteerd door een rij (n, a) in $Student_0$ tezamen met een stel rijen $(n, h_1, r_1), \dots, (n, h_k, r_k)$ in $Student_1$, en er zijn in deze tabellen geen andere rijen dan op deze manier verkregen. Het telnr van een student wordt gerepresenteerd in *Unilid*.

Opmerking (verzamelingsnotatie bekend verondersteld). We kunnen het bedoelde gebruik van de nieuwe tabellen korter formuleren door ze op de volgende manier te relateren aan de oude tabel:

$$Student_0 \text{ join } (Student_1 \text{ group by Naam}) = Student$$

De operatoren *join* en *group by* moeten zó gedefinieerd worden dat, voor dit geval, geldt:

- $(n, a, hs) \in Student_0 \text{ join } (Student_1 \text{ group by Naam}) \Leftrightarrow$
 $(n, a) \in Student_0 \wedge (n, hs) \in (Student_1 \text{ group by Naam})$
- $(n, hs) \in Student_1 \text{ group by Naam} \Leftrightarrow hs = \{h \mid (n, h) \in Student_1\}$

Voor *group by* luidt de formele definitie (enigszins op dit geval toegespitst) als volgt:

- $T \text{ group by } A = \{t:T \bullet (t.A, \{t':T \mid t'.A=t.A \bullet t'.X\})\}$
 waarbij $X =$ alle attributen van T behalve A .

Bijvoorbeeld, $Student_1 \text{ group by Naam} = \{\dots, (n, \{(h_1, r_1), \dots, (h_n, r_k)\}), \dots\}$.

Alle al bestaande verwijzingen naar $Student(Naam)$ moeten nu gewijzigd worden in verwijzingen naar $Student_0(Naam)$. (Bedenk dat een student die géén hobbies heeft wél in $Student_0$ staat en niet in $Student_1$.) Wanneer *Adres* geen attribuut was geweest van $Student$, dat hadden we $Student_0(Naam)$ en $Student_1(Naam, Hobby, Rang)$ gekregen. Tabel $Student_0$ is dan niet overbodig: daarin staan alle bestaande studenten, terwijl in $Student_1$ alleen studenten voorkomen die een hobby hebben.

In algemene termen verwoord hebben we in $Student_0$ alle attributen opgenomen behalve één met gestructureerde waarden: *Hobbies*. In $Student_1$ hebben we juist van dat ene attribuut alle waarde-componenten (namelijk *Hobby* en *Rang*) opgenomen, tezamen met een sleutel van $Student$.

Wanneer, bijvoorbeeld, ook nog *Adres* of *Rang* een gestructureerde waarde heeft, dan gaan we met $Student_0$, respectievelijk $Student_1$, net zo te werk als we nu met $Student$ gedaan hebben. Op deze manier zijn uiteindelijk alle gestructureerde waarden weggewerkt en is het databaseschema in eerste normaalvorm, 1NF.

5 Optimalisatie van het DB-schema

Bij iedere eis in het databaseschema die uit een 1..1-multipliciteitseigenschap bij het ER-diagram is ontstaan, kunnen we twee tabellen samen nemen. In onze casus is daar één voorbeeld van:

Vakgroep en *IsVoorzitter*.

In het ER-diagram heeft relatie *IsVoorzitter* aan de *Prof*-kant de multipliciteit 1..1. Dus in het databaseschema is er bij tabel *IsVoorzitter* een eis dat er bij iedere *Vakgroep*-rij (v, \dots) precies één *Prof*-rij (n, \dots) bestaat met (n, v) in *IsVoorzitter*. Daarom kunnen we *Vakgroep* en *IsVoorzitter*, met al hun eisen, samen nemen en vervangen door deze nieuwe tabel en eisen:

$Vakgroep'(Vakgroepnaam, Gebouw, Naam)$
(Vakgroepnaam) is sleutel.
Naam verwijst naar $Prof(Naam)$.

Het bijhorend gebruik van de tabel is dan als volgt:

Een vakgroep met vakgroepnaam v en gebouw g en als voorzitter een prof met naam n , telnr t en kamer k wordt gerepresenteerd in deze tabel door een rij (v, g, n) ; en er zijn geen andere rijen in de tabel.

We kunnen het bedoelde gebruik van de nieuwe tabel korter formuleren door hem op de volgende manier te relateren aan de oude tabellen:

$Vakgroep' = Vakgroep \text{ join } IsVoorzitter$.

De join operatie werkt als volgt: tabel $Vakgroep'$ bestaat uit de combinaties van een rij van $Vakgroep$ en een rij van $IsVoorzitter$ die gelijke waarden hebben op de gemeenschappelijke attributen (hier: alleen $Vakgroepnaam$). Bijvoorbeeld, een rij (v, g) uit $Vakgroep$ combineert met een rij (n', v') uit $IsVoorzitter$ precies wanneer $v=v'$, en levert dan (v, g, n') in $Vakgroep'$.

We hebben hierboven twee tabellen gecombineerd tot één tabel. Dat is een “verbetering” van het databaseschema in de volgende zin:

- Er is iets minder ruimte nodig om gegevens op te slaan. (Met name: $Vakgroep'$ heeft één kolom minder dan, en altijd evenveel rijen als, $Vakgroep$ en $IsVoorzitter$ samen.)
- Het opzoeken van sommige gegevens gaat iets efficiënter. (Met name: het opzoeken van gebouw én voorzitter van een vakgroep waarvan de vakgroepnaam bekend is gaat in $Vakgroep'$ efficiënter dan in $Vakgroep$ en $IsVoorzitter$ samen.)
- De controle van de 1..1-multipliciteitseigenschap is efficiënter dan wanneer die als een algemene SQL assertie geformuleerd wordt.

Als $IsVoorzitter$ attributen had gehad, dan zouden die ook in $Vakgroep'$ zijn opgenomen. Als $IsVoorzitter$ vele attributen heeft is het (misschien) beter deze optimalisatie achterwege te laten om $Vakgroep$ niet te vertroebelen met die “vakgroepvreemde” attributen.

Discutabele optimalisatie. Bij iedere 0..1 multipliciteit in het ER diagram kunnen we, *wanneer we bereid zijn om NULL toe te laten als waarde in een tabel*, ook weer twee tabellen vervangen door één nieuwe tabel. In ons voorbeeld is er zo'n geval:

- $Student_0$ en $Kiest$. Neem deze samen tot één nieuwe tabel:

$Student'_0(Naam, Adres, Vakgroepnaam, Datum)$
(Naam) is sleutel.
Naam verwijst naar $Unilid(Naam)$.
Vakgroepnaam is $NULL$ of verwijst naar $Vakgroep(Vakgroepnaam)$.
Datum is $NULL$ precies wanneer *Vakgroepnaam* $NULL$ is.

Dankzij het feit dat in SQL $NULL$ is toegestaan als waarde van een foreign key, kan de verwijzingseigenschap in $Student'_0$ geformuleerd worden als:

`foreign key Vakgroepnaam references Vakgroep(Vakgroepnaam).`

Het bijhorend gebruik van de tabel is dan als volgt:

- Een student met naam n , telnr t , adres a , hobbies $\{(h_1, r_1), \dots, (h_k, r_k)\}$ die op datum d een vakgroep kiest met vakgroepnaam v en gebouw g wordt in deze tabel gerepresenteerd door een rij (n, a, v, d) .
- Een student met naam n , telnr t , adres a , hobbies $\{(h_1, r_1), \dots, (h_k, r_k)\}$ die géén vakgroep kiest wordt in deze tabel gerepresenteerd door een rij $(n, a, NULL, NULL)$.

Er zijn geen andere rijen in deze tabel.

In beide gevallen worden het telnr en de hobbies in andere tabellen opgeslagen.

We kunnen het bedoelde gebruik van de nieuwe tabel korter formuleren door hem te op de volgende manier te relateren aan de oude tabellen:

$Student'_0 = Student_0 \text{ leftouterjoin } Kiest$

Deze join-operatie combineert rijen waarvan, voor ieder gemeenschappelijk attribuut, de waarden links en rechts gelijk zijn, en neemt óók nog elke rij van de linker tabel die in de rechter tabel géén overeenkomstige rijen heeft — door de rij met *NULLs* aan te vullen. Bijvoorbeeld, een rij (n, a) uit $Student_0$ combineert met een rij (v, n, d) uit $Kiest$ tot een rij (n, a, v, d) in $Student'_0$; en, wanneer er in $Kiest$ géén rij (v, n', d) is met $n=n'$, dan verschijnt rij (n, a) uit $Student_0$ in $Student'_0$ in de vorm $(n, a, NULL, NULL)$.

In het algemeen verdient het aanbeveling om *NULLs* in tabellen te vermijden, want *NULLs* maken de betekenis van ‘plus’, ‘maal’, ‘=’, ‘and’, ‘or’, ‘count’, avg’ enzovoorts ingewikkelder en geven dus aanleiding tot fouten. Dus het is maar de vraag of de vervanging van $Student_0$ en $Kiest$ (zonder *NULLs*) door één tabel $Student'_0$ (met *NULLs*) de kwaliteit van het databaseschema verbetert!

Het databaseschema dat nu verkregen is (zónder de discutabele optimalisatie) staat getekend in Figuur 4.

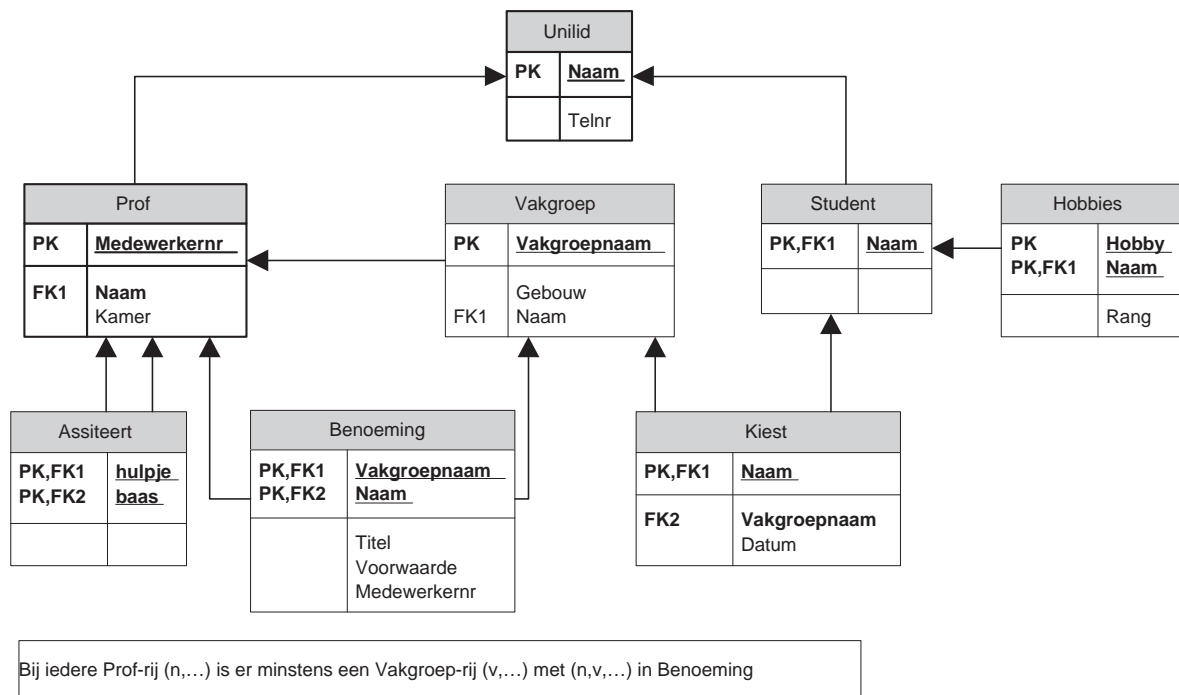
6 Normalisatie tot BCNF

Het kan voorkomen dat er, gemotiveerd vanuit de beoogde toepassing, bij het databaseschema (of —natuurlijk!— al bij het ER-diagram) eigenschappen geëist worden die redundantie en daarmee “anomalieën” veroorzaken. We laten hier zo’n soort eigenschap zien (namelijk: verdachte FDs), en de manier waarop de anomalieën weggewerkt kunnen worden (namelijk: normalisatie tot BCNF). In deze sectie nemen we steeds de universiteitscasus als voorbeeld, met deze student-tabel (waarschijnlijk door een slecht ontwerpproces ontstaan!):

$Student(Naam, Telnr, Adres, Hobby, Rang)$.

Bedoeld gebruik van de tabel: een student met naam n , telnr t , adres a en hobbies $\{(h_1, r_1), \dots, (h_k, r_k)\}$ wordt gerepresenteerd door het stel rijen $(n, t, a, h_1, r_1), \dots, (n, t, a, h_k, r_k)$; en er zijn geen andere rijen in de tabel dan deze.

Dus in deze tabel is $(Naam)$ geen sleutel maar $(Naam, Hobby)$ wel (herinner je dat de rang bepaald wordt door de naam en hobby van een student).



Figuur 4: Grafische weergave van het uiteindelijke databaseschema (zonder de “discutabele” optimalisatie).

Functionele afhankelijkheid, FD. Laat T een tabel zijn. We definiëren relatie \rightarrow (uitgesproken als “bepaalt”) als volgt voor attribootverzamelingen X en Y van T :

$$X \rightarrow Y \Leftrightarrow \text{altijd geldt: rijen van } T \text{ met gelijke waarden voor } X \text{ hebben ook gelijke waarden voor } Y.$$

Bijvoorbeeld, gezien vanuit de universiteitscasus, geldt in *Student*:

- $Naam \rightarrow Telnr, Adres$ “altijd heeft een student hooguit één telnr en adres”,
- $Naam \not\rightarrow Hobby$ “soms heeft een student méér dan één hobby”,
- $Hobby \not\rightarrow Rang$ “soms heeft een hobby méér dan één rang (namelijk, bij verschillende studenten)”.

Merk op dat één tabel-inhoud (= “soms”) een tegenvoorbeeld voor $X \rightarrow Y$ kan zijn, maar dat de formule ‘ $X \rightarrow Y$ ’ zich uitspreekt over *alle* tabelinhouden (= “altijd”). Als $X \rightarrow Y$ geldt, zeggen we ook wel dat de formule ‘ $X \rightarrow Y$ ’ een *functionele afhankelijkheid* is (functional dependency, FD) van T . Die naamgeving komt hier van: wanneer $X \rightarrow Y$ geldt in T , is relatie T in feite een *functie* die bij de X -waarden de Y -waarden oplevert.

Verdachte FDs. Laat T een tabel zijn. Van sommige attribootverzamelingen X en Y van T kunnen we al aangeven dat $X \rightarrow Y$, *onafhankelijk van de vorm, de interpretatie en mogelijke inhouden van T* :

- $X \rightarrow Y$, wanneer X geheel Y omvat.
Bijvoorbeeld, er geldt $Telnr, Adres \rightarrow Telnr$, immers rijen met gelijke waarden voor $Telnr$ én $Adres$ hebben, trivialiter!, gelijke waarden voor $Telnr$.

- $S \rightarrow Y$, wanneer S een sleutel is van T (en Y een willekeurig stel attributen is van T). Inderdaad, een sleutel S is een stel attributen zó dat altijd rijen met gelijke waarden voor S helemaal gelijk zijn. Bijvoorbeeld, $(Naam, Hobby)$ is een sleutel van $Student$, dus: $Naam, Hobby \rightarrow Telnr$.

Ook geldt $S' \rightarrow Y$ wanneer S' méér is dan een sleutel, dat wil zeggen S' omvat een sleutel. Bijvoorbeeld, $Naam, Adres, Hobby \rightarrow Telnr$.

Alle functionele afhankelijkheden die onder bovenstaande twee vormen vallen, noemen we *zeker* en de andere *verdacht* :

$$\begin{aligned} 'X \rightarrow Y' \text{ is zeker in } T &\Leftrightarrow X \text{ omvat } Y \text{ of een sleutel van } T. \\ 'X \rightarrow Y' \text{ is verdacht in } T &\Leftrightarrow 'X \rightarrow Y' \text{ is niet zeker in } T \\ &\Leftrightarrow X \text{ omvat niet geheel } Y, \text{ en geen sleutel van } T. \end{aligned}$$

De termen 'zeker' en 'verdacht' zijn niet algemeen gangbaar; ik heb ze zelf verzonnen. In onze voorbeeldtabel $Student$ zijn ' $Naam, Telnr \rightarrow Telnr$ ' en ' $Naam, Hobby \rightarrow Telnr$ ' zekere FDs, en is ' $Naam \rightarrow Telnr, Adres$ ' een verdachte FD.

Anomalieën. Het is niet moeilijk het volgende in te zien:

- Verdachte FDs veroorzaken redundantie (= dubbele opslag van waarden).

Bijvoorbeeld, omdat $Naam \rightarrow Telnr$ in tabel $Student$, wordt bij iedere student het telnr mogelijk vele malen opgeslagen (namelijk net zo veel keer als de student hobby's heeft).

- Redundantie is ongewenst bij het invoegen, bijwerken en verwijderen van rijen.

Bijvoorbeeld, wanneer het telnr van een student wijzigt, moet dat op *vele* plaatsen in $Student$ aangepast worden; wanneer een student alle hobby's opgeeft, is er *geen* mogelijkheid het telnr op te slaan; wanneer een student toegevoegd wordt, *moet* er ook een hobby,rang-combinatie opgeslagen worden. Deze verschijnselen staan bekend als *anomalieën*, ziekelijke verschijnselen.

Dus, om een kwalitatief goed databaseschema te krijgen moet er iets veranderd worden zó dat de verdachte FDs niet meer verdacht zijn. Een tabel zonder verdachte FDs heet '*in Boyce-Codd Normal Form (BCNF)*' te staan. Het proces om tabellen te krijgen die geen verdachte FDs hebben, heet *normalisatie tot BCNF*.

Voorbeeld van een normalisatie. In $Student$ (met attributen $Naam, Telnr, Adres, Hobby, Rang$) is $(Naam, Hobby)$ de enige sleutel, en geldt: $Naam \rightarrow Telnr, Adres$. Dus ' $Naam \rightarrow Telnr, Adres$ ' is verdacht, en de verdenking willen we elimineren. Daartoe vervangen we $Student$ door de volgende twee tabellen:

- $Student'(Naam, Telnr, Adres)$
Bedoeld gebruik: ... (ligt voor de hand)
 $(Naam)$ is sleutel.
- $Student''(Naam, Hobby, Rang)$
Bedoeld gebruik: ... (ligt voor de hand)
 $(Naam, Hobby)$ is sleutel.

- Het *bedoeld gebruik* kan samengevat worden in deze formule: $Student' \text{ join } Student'' = Student$. Uit een gegeven tabel-inhoud van $Student$ kan de tabel-inhoud van de bedoelde $Student'$ gemaakt worden door alle kolommen te schrappen behalve $Naam, Telnr, Adres$. Analooq voor $Student''$. Er geldt daarna *inderdaad*: $Student' \text{ join } Student'' = Student$.

Eenvoudig is na te gaan dat beide tabellen geen verdachte FDs meer hebben; de normalisatie tot BCNF is in één stap voltooid.

Normalisatie. We formuleren bovenstaand voorbeeld nu in heel algemene termen.

Laat T een tabel zijn en ' $X \rightarrow Y$ ' een verdachte FD. We decomponeren T en passen de representatie van informatie overeenkomstig aan, zó dat de FD niet langer verdacht is:

- Neem aan dat X en Y geen overlap hebben (laat anders de elementen van X weg uit Y). Neem Z zó dat X, Y, Z een partitie is van de attributen van T .
- Vervang $T(X, Y, Z)$ door $T'(X, Y)$ en $T''(X, Z)$, en gebruik deze nieuwe tabellen zó dat:

$$T' \text{ join } T'' = T.$$

(Schrapp de Z -kolommen uit T om T' te krijgen, schrapp de Y -kolommen uit T om T'' te krijgen. Men kan bewijzen dat dan geldt: $T' \text{ join } T'' = T$.)

Merk nu op:

- ' $X \rightarrow Y$ ' is in T' zeker en in T'' geen FD (want Y behoort niet tot de T'' -attributen).
- Eventuele verdachte FDs in T' en T'' zijn al verdacht in T .
(Dit is niet eenvoudig in te zien en vergt een bewijs — hier niet gegeven.)

Dus door deze stap is het aantal verdachte FDs afgenomen.

Door de stap herhaaldelijk te doen op tabellen waarvoor er nog verdachte FDs zijn, ontstaat er een decompositie T_1, \dots, T_n van T die geen verdachte FDs heeft en dus in BCNF staat. Daarmee is de redundantie die veroorzaakt werd door verdachte FDs, weggewerkt terwijl T zelf gereconstrueerd kan worden door $T = T_1 \text{ join } \dots \text{ join } T_n$.

Berekening van FDs. Bovenstaand normalisatie-proces vervangt T door T' en T'' , en vraagt dan om T' en T'' op gelijke wijze verder te normaliseren. Maar als er in T een stel FDs bekend zijn, wat zijn dan van de nieuwe T' de FDs? Die heb je nodig om de sleutel en verdachte FDs van die nieuwe tabel te bepalen!

Het volgende is gemakkelijk in te zien, voor een T' die uit T ontstaat door kolommen weg te laten:

Voor attribuutverzamelingen X, Y van T' geldt: $X \rightarrow Y$ in $T' \iff X \rightarrow Y$ in T .

Dus resteert alleen nog de vraag: welke FDs zijn er in T als er een stel FDs *gegeven* is? Bijvoorbeeld, wanneer in T is gegeven: $A \rightarrow B$ en $B \rightarrow C$, dan volgt uit de definitie van ' \rightarrow ' dat ook $A \rightarrow C$. Het volgende algoritme laat zien hoe je bij gegeven FDs in T en gegeven X , de grootst mogelijke Y kan vinden waarvoor geldt: $X \rightarrow Y$.

Neem Y initieel gelijk aan X .

Herhaal nu de volgende stap totdat Y niet meer aangroeit:

voor iedere $U \rightarrow V$ die *gegeven* is in T met $U \subseteq Y$, voeg je V toe aan Y .

Het is makkelijk in te zien dat voor de aldus berekende Y geldt: $X \rightarrow Y$. Het vergt enige moeite om te bewijzen (hier overgeslagen) dat je zo de grootst mogelijke Y krijgt met $X \rightarrow Y$.

Voorbeeld. Bij gegeven $A \rightarrow B$ en $B \rightarrow C$ berekenen we de grootst mogelijke Y met $A \rightarrow Y$:

Neem Y initieel gelijk aan A , dus $Y_0 = A$.

Op grond van de gegeven ' $A \rightarrow B$ ' breiden we Y uit met B , dus $Y_1 = AB$.

Op grond van de gegeven ' $B \rightarrow C$ ' breiden we Y uit met C , dus $Y_2 = ABC$.

De gegeven FDs geven geen verdere uitbreiding van Y . Het antwoord is: $Y = ABC$.

Nog een voorbeeld van FDs. Stel dat we voor de universiteitscasus een databaseschema hebben met onder andere de volgende tabel:

ProfPromVakgrp(Naam, Medewerkernr, Kamer, Vakgroepnaam, Gebouw)

Bedoeld gebruik van de tabel: een prof met naam n , telnr t , medewerkernr m , kamer k , en een vakgroep met vakgroepnaam v en gebouw g worden in deze tabel gerepresenteerd door een rij (n, m, k, v, g) ; en er zijn geen andere rijen in de tabel dan op deze manier verkregen.

Dan zijn $(Naam, Vakgroepnaam)$ en $(Medewerkernr, Vakgroepnaam)$ de enige sleutels, en zijn er de volgende verdachte FDs:

$Naam \rightarrow Medewerkernr, Kamer$

$Medewerkernr \rightarrow Naam, Kamer$

$Vakgroepnaam \rightarrow Gebouw$

Vaak duidt de aanwezigheid van verdachte FDs er op dat er té veel ongerelateerde informatie in één tabel wordt opgeslagen. In de tabel hierboven worden zowel profs als ook vakgroepen opgeslagen, en dat kan beter (= met minder redundantie en anomalieën) in twee tabellen gebeuren.

Opmerking. Bij normalisatie tot BCNF kan het zo zijn dat een gegeven FD niet meer te formuleren is voor de resulterende tabellen. Als we de controle van de eis, uitgedrukt door de FD, willen laten uitvoeren door het databasesysteem, dan moet het systeem eerst de oorspronkelijke tabel reconstrueren, alvorens de controle te kunnen uitvoeren. Dat is een dure zaak. Een alternatief is om niet tot BCNF te normaliseren (en daarmee alle FD-redundantie weg te werken), maar iets minder redundantie weg te werken. Dat kan door slechts tot de derde normaalvorm te normaliseren. We leggen dat hier niet verder uit.

7 Normalisatie tot 4NF

We beschouwen een variant van de universiteitscasus waarbij een student niet alleen verscheidene gerangschikte hobby's heeft, maar ook verscheidene telnr's. Beschouw de volgende tabel (waarschijnlijk door een slecht ontwerpproces ontstaan!) ter representatie van de informatie:

$Student(Naam, Telnr, Adres, Hobby, Rang)$.

- $(Naam, Telnr, Hobby)$ is de enige sleutel.
- Bedoeld gebruik van de tabel: een student met naam n , telnr's $\{t_1, \dots\}$, adres a en hobbies $\{(h_1, r_1), \dots\}$ wordt gerepresenteerd door het stel rijen $(n, t_1, a, h_1, r_1), \dots, (n, t_i, a, h_j, r_j), \dots$; en er zijn geen andere rijen in de tabel dan deze.

Voor deze tabel zijn 'Naam \rightarrow Adres' en 'Naam, Hobby \rightarrow Rang' verdachte FDs. Normalisatie tot BCNF levert drie tabellen:

$Student_1(Naam, Adres)$,
 $Student_2(Naam, Hobby, Rang)$, en
 $Student_3(Naam, Telnr, Hobby)$.

Ondanks het feit dat $Student_3$ in BCNF staat, is er voor die tabel bij het bedoelde gebruik ervan redundantie (met de daaruit voortvloeiende anomalieën): zowel telnr's als ook hobby's worden veelvuldig opgeslagen. Deze redundantie gaan we wegwerken.

We observeren het volgende over het gebruik van $Student_3$:

Als (n, t_1, h_1) en (n, t_2, h_2) in de tabel voorkomen,
dan ook (n, t_1, h_2) en (n, t_2, h_1) .

Informeel gezegd komt het er op neer dat $Telnr$ meerwaardig is, en ook $Hobby$ meerwaardig is: iedere waarde voor de één combineert met iedere waarde voor de ander. Zo'n eigenschap heet *multivalued dependency* (MVD) en wordt in dit geval symbolisch genoteerd als volgt:

$(Naam, Telnr, Hobby) = (Naam, Telnr) \text{ join } (Naam, Hobby)$.

Deze notatie suggereert dat iedere waarde voor $Telnr$ combineert met iedere waarde voor $Hobby$. Merk op dat join een operatie is op tabellen, maar in deze *symbolische* notatie gebruikt wordt *als of het een operatie is op attribuutverzamelingen!* Een tabel waarvoor er geen verdachte MVD is, heet in *4e normaalvorm* te staan, 4NF.

Normalisatie. Laat T een tabel zijn en 'XYZ = XY join XZ' een MVD voor T , waarbij X, Y, Z een partitie vormt van de attribuutverzameling van T . De decompositie van T voor deze MVD gaat exact het zelfde als de decompositie van T voor FD 'X \rightarrow Y', en beschrijven we daarom maar beknopt:

Vervang $T(X, Y, Z)$ door $T'(X, Y)$ en $T''(X, Z)$, en gebruik deze nieuwe tabellen zó dat:

$$T' \text{ join } T'' = T.$$

De MVD is niet meer aanwezig in T' en T'' , en T kan gereconstrueerd worden door $T = T' \text{ join } T''$. Zo nodig moet deze stap herhaald worden op de nieuw verkregen tabellen.

(Men kan eenvoudig bewijzen dat in een tabel $T(XYZ)$ iedere FD 'X \rightarrow Y' tevens een MVD 'XYZ = XY join XZ' is. Dus een schema in 4NF is ook in BCNF.)

Voorbeeld.

De decompositie van $Student_3$ voor de MVD ‘ $(Naam, Telnr, Hobby) = (Naam, Telnr) \underline{join} (Naam, Hobby)$ ’ levert:

$Student_{3,1}(Naam, Telnr)$ en $Student_{3,2}(Naam, Hobby)$.

Bedoeld gebruik: ... (ligt wel voor de hand).

De tabel $Student$ waarmee we deze sectie begonnen, is dus tot BCNF en zelfs tot 4NF genormaliseerd met als resultaat: $Student_1$, $Student_2$, $Student_{3,1}$ en $Student_{3,2}$. Dankzij de gevolgde methode is er geen informatie verloren gegaan en geldt er:

$Student = Student_1 \underline{join} Student_2 \underline{join} Student_{3,1} \underline{join} Student_{3,2}$.

Opmerking. Het voorbeeld van deze sectie is enigszins misleidend omdat in de MVD ‘ $XYZ = XY \underline{join} XZ$ ’ zowel de Y ($Telnr$) als ook de Z ($Hobby$) uit slechts één attribuut bestaat, terwijl in het algemeen de X , Y en Z ieder uit een stel attributen mogen bestaan. Een voorbeeld met een Y die uit twee attributen bestaat is deze: vervang $Telnr$ door $(Netnr, Abonneenr)$. Een voorbeeld met een Z die uit twee attributen bestaat is deze: vervang $Hobby$ door $(Activiteit, Gereedschap)$.

MVDs voorkómen. Wanneer meerwaardige attributen vroegtijdig onderkend worden en als attributen met gestructureerde waarden worden behandeld, dan zullen er met onze wijze van database-ontwerp geen MVDs ontstaan, maar moeten we in ruil daarvoor een (gemakkelijke!) normalisatie-stap tot 1NF uitvoeren om de gestructureerde waarde-verzamelingen uit het databaseschema weg te werken.

8 Alternatieven voor generalisatie/specialisatie

Voor $Unilid$ met specialisaties $Prof$ en $Student$ zijn er (minstens) drie alternatieven voor de vertaling van het oorspronkelijke ER-diagram naar een databaseschema. Deze luiden als volgt — de eerste hebben we hierboven gekozen. Korthedshalve laten we steeds de declaratie van de sleutel(s) weg.

- Representeer in een $Unilid$ -tabel de gegevens die alle unileden gemeenschappelijk hebben, in een $Prof$ -tabel alleen de gegevens (en een sleutel) die specifiek zijn voor een prof, en in een $Student$ -tabel alleen de gegevens (en een sleutel) die specifiek zijn voor een student. $Telnr$ komt dus alleen in $Unilid$:
 - $Unilid(Naam, Telnr)$,
 - $Prof(Naam, Medewerkernr, Kamer)$,
 - $Student(Naam, Adres, Hobbies : \underline{set}(Hobby, Rang))$.
- Beoogd gebruik: ... (ligt voor de hand) met in het bijzonder:
 - $Prof(Naam)$ verwijst naar $Unilid(Naam)$
 - $Student(Naam)$ verwijst naar $Unilid(Naam)$
- Representeer in $Prof$ alle prof-gegevens, in $Student$ alle student-gegevens en in $Unilid$ alléén de unileden die niet prof en niet student zijn. $Telnr$ komt nu dus in alle tabellen:

- *Unilid*(*Naam*, *Telnr*),
Prof(*Naam*, *Telnr*, *Medewerkernr*, *Kamer*),
Student(*Naam*, *Telnr*, *Adres*, *Hobbies* : set(*Hobby*, *Rang*)).
 Beoogd gebruik: ... (ligt voor de hand) met in het bijzonder:
Prof(*Naam*) komt *NIET* voor als *Unilid*(*Naam*)
Student(*Naam*) komt *NIET* voor als *Unilid*(*Naam*)
 - Tabel *Unilid* kan zelfs weggelaten worden wanneer ieder unilid prof is of student of beide, dat wil zeggen de generalisatie is *covering*.
 - Gemeenschappelijke gegevens (*Telnr*) worden gedupliceerd opgeslagen wanneer een unilid zowel prof als ook student kan zijn (dat wil zeggen, de generalisatie is niet *disjunct*). Duplicatie van gegevens is vervelend (zoals op bladzijde 15 wordt uitgelegd). Dus wanneer een generalisatie niet *disjunct* is, is deze methode af te raden.
 - Voor- en nadeel van deze methode. Als de generalisatie *disjunct* is, is er nu iets minder opslagruimte nodig dan bij de eerste methode. Het opzoeken van een telnr van een prof gaat nu iets efficiënter dan bij de eerste methode, en het opsommen van alle unileden iets minder efficiënt.
- Repreenteer alle gegevens in één tabel. Er zijn nu dus geen aparte tabellen voor *Prof* en *Student*:
 - *Unilid*(*Naam*, *Telnr*, *Medewerkernr*, *Kamer*, *Adres*, *Hobbies* : set(*Hobby*, *Rang*))
 - Beoogd gebruik: ... (ligt voor de hand) met in het bijzonder:
Medewerker≠*NULL*≠*Kamer* precies wanneer het unilid een prof is.
Adres≠*NULL*≠*Hobbies* precies wanneer het unilid een student is.
 - Voor- en nadeel van deze methode. Er zijn *NULLs* nodig; dat is een nadeel (want *NULLs* maken de betekenis van ‘plus’, ‘maal’, ‘=’, ‘and’, ‘or’, ‘count’, ‘avg’ enzovoorts ingewikkelder en geven dus aanleiding tot fouten). Het opsommen van alle unileden met al hun gegevens gaat iets efficiënter dan bij de vorige methoden, het opsommen van precies de profs minder efficiënt.

De beoogde toepassing bepaalt welke van deze drie methoden de voorkeur heeft.

9 Meer eigenschappen bij het ER-diagram

Het ER-diagram is een beschrijving van de werkelijkheid, gezien vanuit het perspectief van de bedoelde toepassingen. Aan de hand van het ER-diagram wordt het databaseschema ontworpen. Dus het ER-diagram moet zo goed mogelijk zijn: iedere eigenschap in de werkelijkheid die van belang is voor de toepassing, zou —idealiter— in of bij het ER-diagram genoteerd moeten worden. Naast de eigenschappen die het *bestaan* van entiteiten en relaties aangeven, hebben we tot nu toe deze soorten eigenschappen gezien:

- multipliciteit-, generalisatie- en sleutel-eigenschappen;
- de *covering* en *disjoint* bij een generalisatie/specialisatie;
- functionele afhankelijkheden (FDs);
- meerwaardigheids-afhankelijkheden (MVDs).

Daarnaast kun je ook nog denken aan eigenschappen zoals deze:

per student hebben verschillende hobby's ook verschillende rangs.

Ook zijn *domein*-eigenschappen denkbaar, zoals deze:

een rang bestaat uit een positief geheel getal,
een telnr bestaat uit een reeks van 10 cijfers,
een naam bestaat uit hooguit 32 letters en spaties-tussen-de-letters.

Iedere eigenschap die in of bij het ER-diagram genoteerd staat, verschijnt na vertaling als een —te eisen— eigenschap bij het databaseschema. Met zo'n eis kunnen we als volgt omgaan:

- We laten de eigenschap bij iedere wijziging van de database-inhoud controleren door het databasesysteem. In dit geval komt zo'n eis bijvoorbeeld te voorschijn in een tabel- of domein-definitie als een *CHECK*, of bij de gehele database als een *ASSERTION*. (De *CHECK* en *ASSERTION* komen bij de behandeling van SQL aan bod.)
- We laten de controle van de eigenschap achterwege, in de veronderstelling dat de gebruiker of toepassingsprogrammatuur van de database zich aan de eis houdt.

We zullen in de rest van deze sectie een soort eigenschap illustreren die nog niet aan bod is gekomen: de cykel-eigenschap.

Cykel-eigenschappen. Een *cykel* in het ER-diagram is een reeks entiteitstypen (de laatste gelijk aan de eerste) die door relaties of generalisatie/specialisatie met elkaar verbonden zijn. In het voorbeeld ER-diagram zijn de volgende “elementaire” cycli te onderscheiden:

Prof *Assisteert* *Prof*
Prof *IsVoorzitter* *Vakgroep* *Benoeming* *Prof*
Prof \triangleright *Unilid* \triangleleft *Student* *Kiest* *Vakgroep* *IsVoorzitter* *Prof*
Prof \triangleright *Unilid* \triangleleft *Student* *Kiest* *Vakgroep* *Benoeming* *Prof*

Daarnaast kun je deze cycli ook aan elkaar plakken, om “samengestelde” cycli te krijgen. Bijvoorbeeld: de eerste twee genoemde cycli leveren, aan elkaar geplakt, deze cykel op:

Prof *Assisteert* *Prof* *IsVoorzitter* *Vakgroep* *Benoeming* *Prof*

De ervaring leert dat bij bijna iedere elementaire cykel er één of meer eigenschappen in de werkelijkheid zijn aan te wijzen die bij het ER-diagram opgenomen moeten worden om een ‘geschikt’ diagram te krijgen. Hier zijn een paar voorbeelden van zulke eigenschappen; sommigen zijn zeker niet waar in werkelijkheid (en moeten niet als eis in het ER-diagram worden vermeld) en anderen misschien wel:

- Bij de cykel *Prof* *Assisteert* *Prof*:
Als prof h assistent is van prof b , dan is b niet assistent van h .
Als prof h assistent is van prof b , dan is $h \neq b$.
Als p_0 is assistent van p_1 is assistent van $p_2 \dots$ is assistent van p_{k+1} , dan $p_0 \neq p_{k+1}$.
(Deze laatste eigenschap impliceert de twee voorgaande.)
- Bij de cykel *Prof* *IsVoorzitter* *Vakgroep* *Benoeming* *Prof*:
Als prof p voorzitter is van vakgroep v , dan heeft p een benoeming in v .
Als prof p een benoeming heeft in vakgroep v , dan is p voorzitter van v .

- Bij de cykel $Prof \triangleright Unilid \triangleleft Student \underline{Kiest} \underline{Vakgroep} \underline{IsVoorzitter} Prof$:
 Als prof p ook student is en vakgroep v kiest, dan is p voorzitter van v .
 Als prof p ook student is en vakgroep v voorzit, dan kiest p vakgroep v .
- Bij de cykel $Prof \triangleright Unilid \triangleleft Student \underline{Kiest} \underline{Vakgroep} \underline{Benoeming} Prof$:
 Als prof p een benoeming heeft in vakgroep v en p is student, dan kiest p vakgroep v .

De cycli via de generalisatie zijn nietszeggend, wanneer de generalisatie *disjoint* is.

Nogmaals: Vanuit het perspectief van de bedoelde toepassing moet beoordeeld worden of dergelijke cykel-eigenschappen correct en van belang zijn, en of we de controle daarvan willen laten uitvoeren door het databasesysteem (bij iedere wijziging van de database-inhoud) of willen overlaten aan de gebruikers van de database.

Gebruikte technische termen

1NF 4NF BCNF Boyce-Codd DB-schema ER-diagram ER-modellering FD-redundantie FDs KEY MVD SQL UML UML-notatie “elementaire” ‘correct’ ‘fout’ ‘geschikt’ ‘leesrichting’ afhankelijkheden afhankelijkheid anomalieën assertie attributen attribuut attribuutverzameling attribuutwaarden correctheid cykel cykel-eigenschap database database-inhoud database-ontwerp databaseschema databasesysteem decomponeren decompositie default defaultwaarde dependency diagram diagram-instantie disjoint domein-definitie domein-eigenschappen duplicatie entiteit entiteitstype form functional functionele generalisatie generalisatie/specialisatie generalisatie/specialisatie-eigenschappen generalisatie/specialisatie-knoop genormaliseerd gestructureerd groupby incorrect incorrectheid instantie join join-operatie kolommen leesrichting leftouterjoin meerwaardig meerwaardigheids-afhankelijkheden multipliciteit multipliciteitseigenschap multivalued normaalvorm normalisatie normalisatie-proces normalisatie-stap normaliseren optimalisatie optimaliseren redundantie relatie relatie-lijn representatie representatiekeuze representeren rij rol rolnaam schema semantiek sleutel sleutel-eigenschappen specialisatie tabel tabelinhouden tabelinstantie tabelschema verdacht verzameling vulling waarde-componenten waarde-verzamelingen waarden