DATABANKEN

GILLES CALLEBAUT

INLEIDING

EERSTE DEFINIETIE VAN DATABASE

Een logische coherente verzameling van gerelateerde gegevens (data), ontworpen voor een specifiek doel en opgeslagen op een "drager".

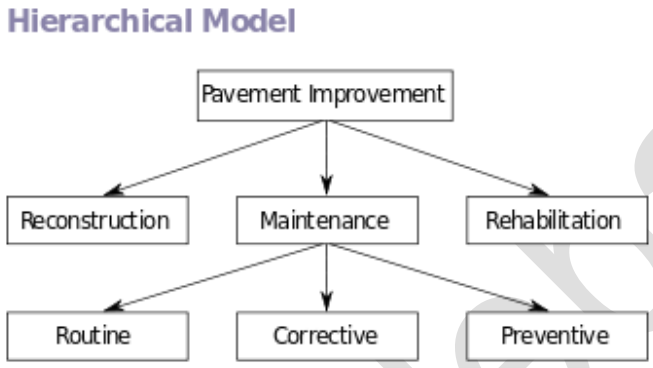
File-niveau: adding & removing
Data-niveau: inserting, retrieving, updating deleting

STRATEGIE

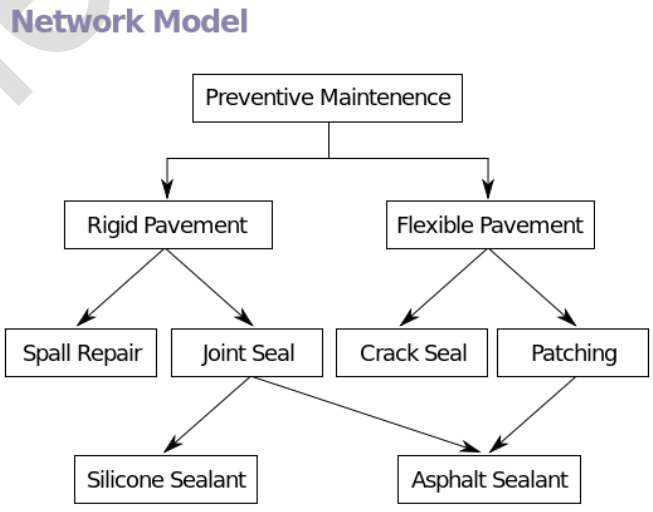
- Modelleren:
  inzicht krijgen in wat de gebruikers willen, wat resulteert in het **gegevensmodel** (Entiteit/Relatie model)
- Ontwerpen (van de relationele database):
  Ontwerp die zo'n accurate mogelijke weergave is van het model
- Implementatie

SOORTEN DATABANKEN

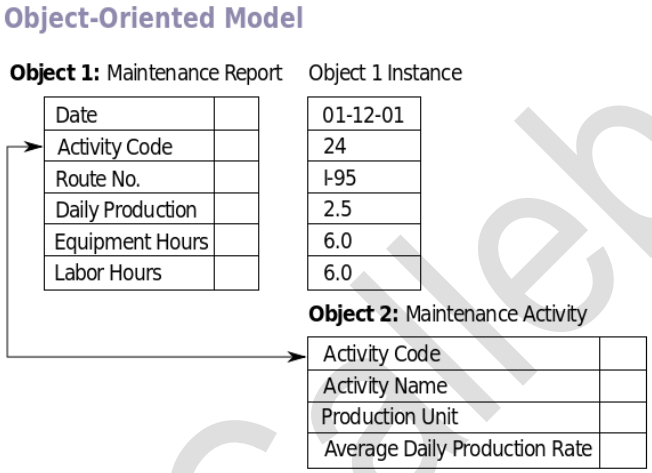
- Hiërarchische databank (1:N)
  De hiërarchische database gaat ervan uit dat **elk record** in een database weer **kan verwijzen naar een n-aantal andere records**. Het is zo een boomstructuur, die steeds verder kan vertakken. Kenmerkend is wel dat ieder recordtype één en niet meer dan één eigenaar (owner) kent.
  Het hiërarchische model kent maar één boom per database, **de takken hebben onderling geen samenhang** en de enige ingang van de boomstructuur is van bovenaf.



- Netwerk databanken (N:M)
  Het breidt de mogelijkheden van gegevensopslag van de eerste generatie databases, de hiërarchische databases, uit met de **mogelijkheden van meerdere paden**, wat impliceert dat de parent-child-relatie tussen records niet meer 1:n is, maar **n:m**. Ieder child kan dus meerdere parents hebben.
  De mogelijkheden van de netwerkdatabase zijn groter dan die van de hiërarchische database, maar blijven beperkt. 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 **database pointers corrupt raken**, dan kunnen **hele delen** van de database opeens **niet meer bereikbaar** zijn.
  De vrijheid in de hoeveelheid parent-child-relaties maakt ook dat het ontwerp van een netwerkdatabase enorm **complex kan worden**.



- Objectgeoriënteerde databank
  Objectgeoriënteerde database model is een databasemodel waarin wordt gewerkt met objecten, net zoals in objectgeoriënteerde programmeertalen.
  Het doel van zo'n database is het invoegen van dergelijke objecten in de database zo eenvoudig te maken. Zo wenst men de objecten die men gebruikt in een objectgeoriënteerde taal direct te kunnen opslaan in de database, zonder transitie naar tupels[1] zoals in een relationele database.
  Men kan **data sneller opvragen** doordat er **geen join operaties nodig zijn**, men kan de pointers rechtstreeks volgen. Dit is dan een voordeel ten opzichte van relationele databases.



DOELSTELLING

Gebruik maken van **SQL** om **toegang te krijgen** tot een databank en **bewerkingen uit te voeren** in een databank.

[1] Formeel is een tupel een element van een eindig Cartesisch product.
Een voorbeeld van een tupel is: ("Dorpsstraat", 123, "Jan Janssen", 38) met straatnaam (een string), huisnummer (een natuurlijk getal), naam (ook een string) en leeftijd (ook een natuurlijk getal). Dit is een ander tupel dan ("Dorpsstraat", 123, 38, "Jan Janssen") aangezien deze in een andere volgorde staan.

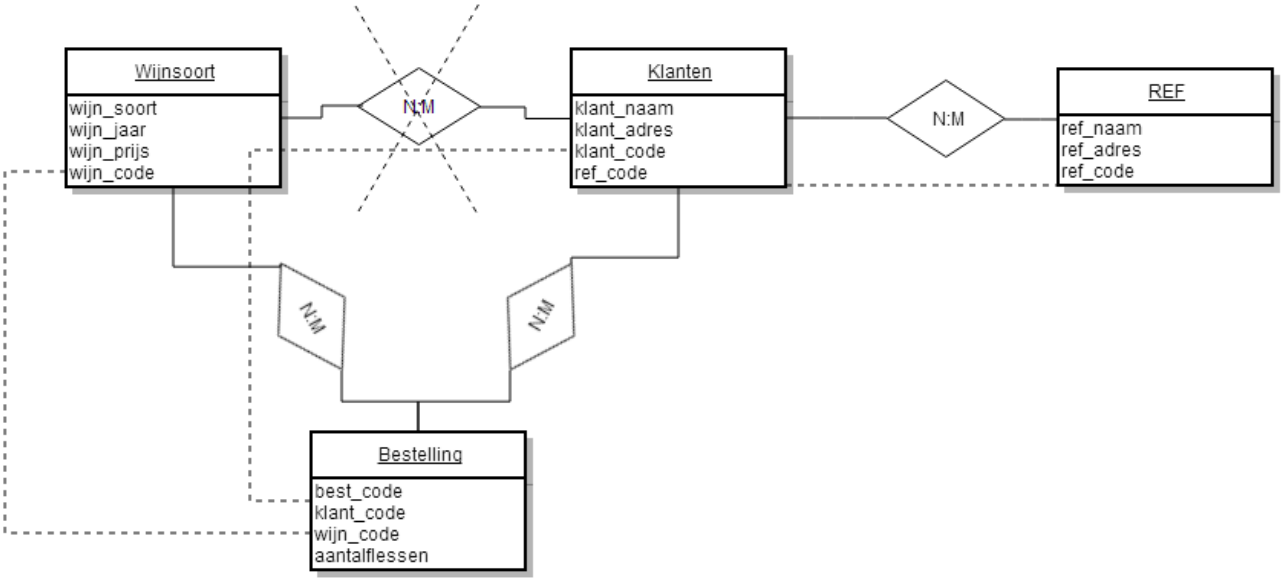
ALGEMENE BEGRIPPEN I.V.M. DATABASE-VERWERKING

(RELATIONELE) TABEL (FILE)

Een **entiteit** is iets waarover men **informatie** wil **opslaan**.
De gegevens hebben verwantschap met elkaar –vormen een **relatie**-, en worden in ene relationele databank opgeslagen in een **tabel**.

Een **rij** van zo'n **tabel** is een **record** of **tupel**.
Een **kolom** van zo'n **tabel** is een **veld** of **attribuut**.

Verschil tussen data en informatie:
Data is de waarden en informatie is de betekenis achter die waarden.



Een **veel-op-veel relatie** geeft aanleiding tot redundantie, dubbele gegevens, veel velden,… de oplossing hiervoor is een **tussentabel** te **creëren**.
Waarbij gebruik wordt gemaakt van sleutels.

CONSTRUCTIE DATABANK

- Definiëren databank
  vastleggen recordstructuren van iedere file, tabel, datatypes en constraints (opgelegde beperkingen)
- Construeren databank (data opgeslagen)
- Manipuleren databank (quering, updating, rapporteren, generen)

MULTI-USER GEBRUIK

Inconsistent read:

Als er **tussen 2 opdrachten** de **informatie** al werd **aangepast**.

Transaction 1

```
/* Query 1 */
SELECT * FROM users WHERE id = 1;
```

Transaction 2

```
/* Query 2 */
UPDATE users SET age = 21 WHERE id = 1;
COMMIT; /* in multiversion concurrency
   control, or lock-based READ COMMITTED */
```

```
/* Query 1 */
SELECT * FROM users WHERE id = 1;
COMMIT; /* lock-based REPEATABLE READ */
```

Phantom read:

Als aangepaste **gegevens niet werden bevestigd**.
Via een **rollback** moet men terug gaan naar de vorige staat.

Transaction 1

```
/* Query 1 */
SELECT * FROM users
WHERE age BETWEEN 10 AND 30;
```

Transaction 2

```
/* Query 2 */
INSERT INTO users VALUES ( 3, 'Bob', 27 );
COMMIT;
```

```
/* Query 1 */
SELECT * FROM users
WHERE age BETWEEN 10 AND 30;
COMMIT;
```

Lost update problem**:**

Als er een **update** gebeurde **nadat de oude gegevens** al door een ander werden gelezen.

SELECT aantalVaten
FROM STOCK
WHERE item=’Duvel’
Bereken 120 -10 =110

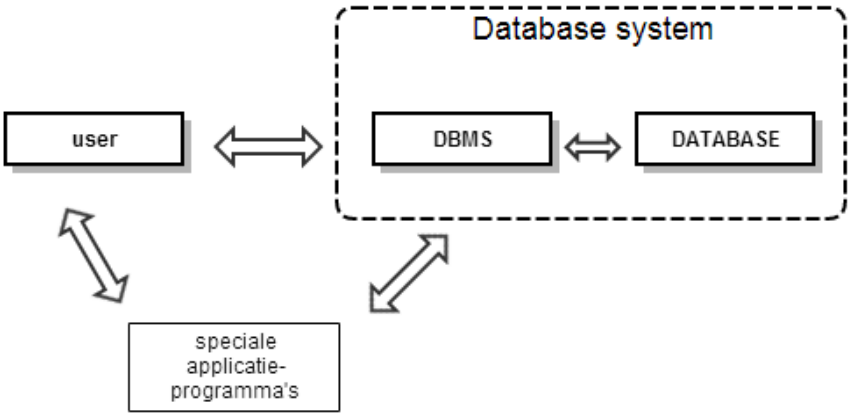
UPDATE STOCK
SET aantalVaten=110
WHERE item=’Duvel’

SELECT aantalVaten
FROM STOCK
WHERE item=’Duvel’
Bereken 120-30=90

UPDATE STOCK
SET aantalVaten=90
WHERE item=’Duvel’

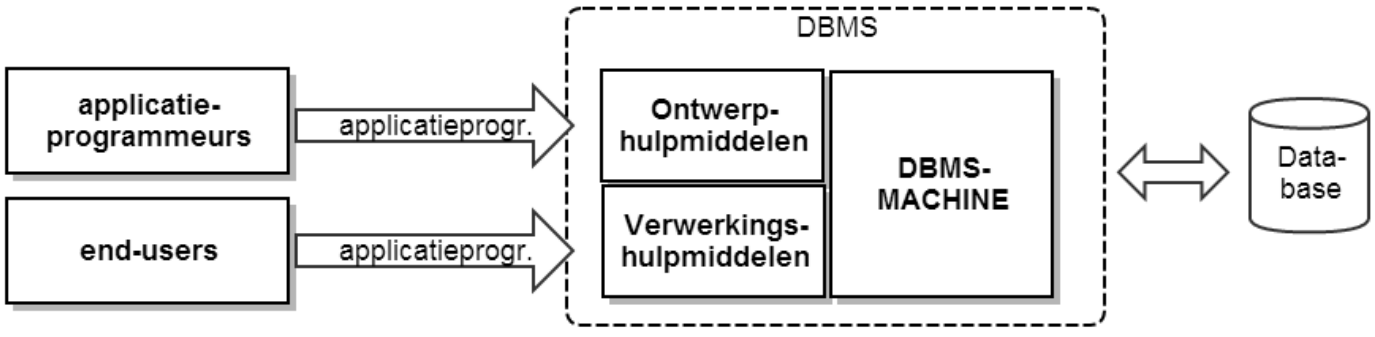
HET DATABASE SYSTEEM

De databank wordt **gecreëerd en gemanipuleerd via het DBMS** (=DataBase Management System).



4 COMPONENTEN VAN DATABASE

- Data
  - Relatietabellen (gebruikersgegevens)
  - Meta-data (beschrijving van eigen structuur)
  - Indexen (toegankelijkheid en verwerkingssnelheid verhogen)
  - Applicatie meta-data (bv. Gegevens over de opmaak van formulieren)
- Hardware
- Software
  - DBMS is de software dat acces tot de databank afhandelt.
    Hoofdfunctie: **user-interface** tot databank
    = grens waardoor alles onzichtbaar is voor gebruiker.
    Componenten DBMS:
    - Ontwerpgereedschappen (ontwerpen en construeren)
    - Run-time programmatuur (vb. query-verwerking)
    - DBMS-machine (verwerken,
      vb. aanvragen voor gegevens vertalen voor het lezen/schrijven op harde schijf)
- Users
  - End-users
  - Applicatieprogrammeurs
  - Database admin



HET DBMS

Collection of programs that enables users to create and maintain a database.

Vergemakkelijken van processen:

- **Defining** (structure, types, constraints)
- **Constructing** (storage)
- **Manipulating** (query)

DE EVOLUTIE VAN BESTANDSVERWERKINGSSYSTEEM NAAR DATABASEVERWERKINGSSYSTEEM

Bestandsverwerkingssysteem = File-processing
Groepen records in afzonderlijke bestanden opgeslagen.

Bij File-processing definieert en implementeert iedere gebruiker de bestanden, nodig voor een specifieke toepassing als deel van de toepassing.

Beperking:

- Gescheiden en geïsoleerde gegevens
- Gegevens worden vaak gedupliceerd
- Applicatieprogramma's afhankelijk van gebruikte file-structuur
- Incompatibele bestanden (geschreven in een andere taal)

Oplossing:
Database verwerkingssysteem

- Data-independence
- Definitie en beschrijving van database structuur en constraints opgeslagen in system catalog (=meta-data)

Voorbeeld bioloog vs zoöloog, verschillende externe niveaus, maar 1 concept niveau en 1 internal level.

DE 3 BELANGRIJKSTE KARAKTRISTIEKEN VAN EEN DATABANK

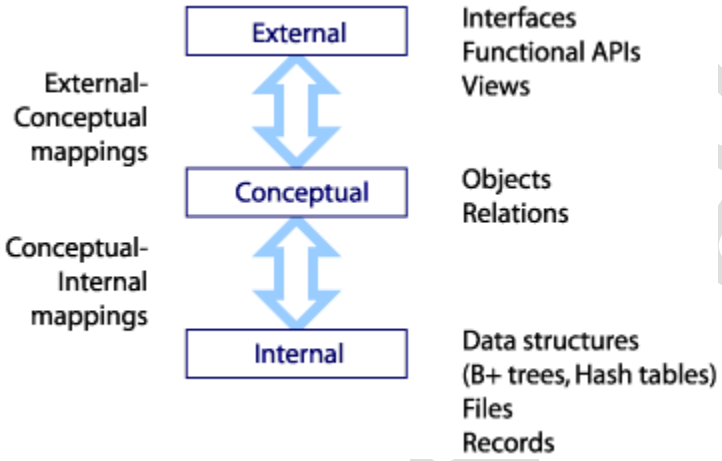
- **Isoleren** van data en programma's
  (fysische databank en de user applications)
- **Ondersteunen** van **verschillende user views** (zie vb. vissen in rivieren)
- Een **cataloog** (meta-data) om de **database beschrijving** te stockeren.
  DB architectuur beïnvloeden.

DATABASE ARCHITECTUUR (ANSI/SPARC)

ANSI American National Standards Institute
SPARC Systems Planning & Requirements Committee

3-LAGEN STRUCTUUR

- **External level** (verschillende views)
  external view; **hoe** wordt de **data "gezien"**.
- **Conceptual level** (1 view)
  structuur van de gehele **databank**
  Bijschrijving, entiteiten, datatypes, relaties, user operations, de constraints;
  het opstellen van dit schema gebeurt bij het initieel ontwerp van de database.
- **Internal level**
  Hoe data (fysisch) **opgeslagen** is.



Mapping
The processes of transforming requests and results between levels.

DDL:
data definition language
de definitie/ declaratie van data-objects

DML:
data manipulation language
manipulatie of processing van data-objects

De “**storage definition language**” voor de **internal level** wordt gewoonlijk apart gehouden en zit dus niet in SQL.

SQL is dus een high end level DML.
Er kunnen dus meerdere records tegelijk behandeld worden in 1 DML statement.

Taak van de database administrator:

- Definiëren van het conceptuele schema:
  Welke info wordt opgeslagen.
  gebruik van conceptuele DLL
- Bepalen van het “internal schema”:
  Beschrijving van de fysische stockage.

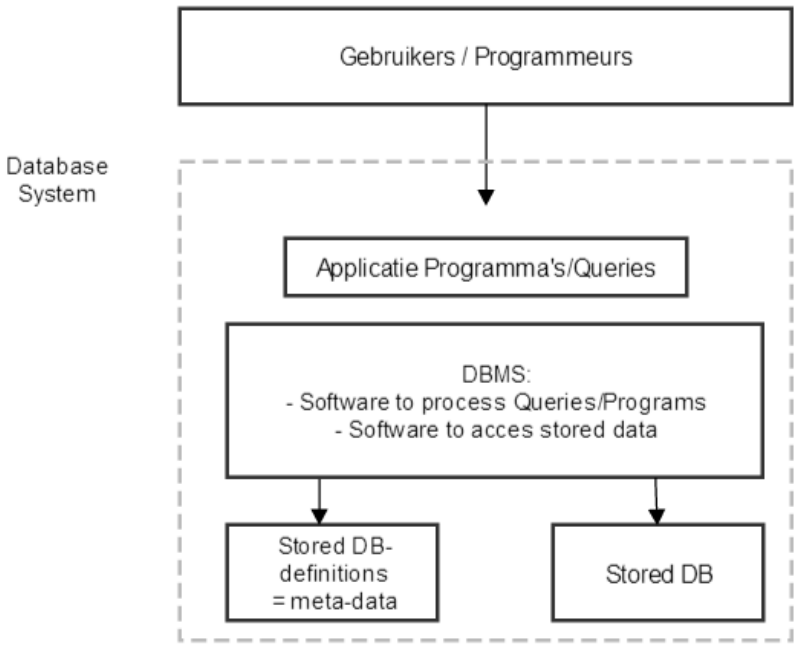
WERKING VAN EEN DBMS

Alle acces behandelt:

- De user verzoekt om acces
- Het DBMS intercepteert dat verzoek en analyseert het
- Het DBMS onderzoekt het “external schema” voor die user
- Het DBMS onderzoekt het “external/conceptual” mapping
- Het DBMS onderzoekt het “external schema”
- Het DBMS onderzoekt de “concetpual/internal” mapping
- Het DBMS onderzoekt het “storage-structure” definitie
- Het DBMS voert de nodige operaties uit op de databank

Functies van het DBMS:

- Het behandelen van de data-definities[2].
  Het DBMS moet “language proccesor” componenten bevatten van de verschillende DLL-talen en de data-definities kunnen interpreteren en linken met de onderliggende niveaus.
- Data manipulatie (d.m.v. een DML-processor)
  verzoek users kunnen behandelen
- Data security, gegevensintegriteit, data recovery
  Data dictionary functie voorzien:
  Metadata;
  Alle schema’s en mappings (data-definities dus) moeten fysisch opgeslagen worden in bron- en objectvorm in die bibliotheek, alsook andere informatie. (beschrijving databank)



[2] (external, conceptual, internal schema, mappings)

NIEUWE DEFINITIE DATABANK

Een database is een zichzelf beschrijvende verzameling van geïntegreerde records.
Dus zowel gebruik gegevens MAAR OOK een beschrijving van zijn eigen structuur (data-dictionary).

Belang van data-dictionary:

- Data independence
  Programma/gegevensonafhankelijkheid wordt vergroot
  immuniteit van applicaties voor veranderingen in de manier waarop de gegevens gestockeerd en ge-acces worden
  Externe documentatie zoals bestandsverwerkingssystemen is niet nog want de structuur kan uit de databank zelf gehaald worden
- Bij wijzigingen van de gegevensstructuur in de databank moeten die veranderingen enkel in de data-dictionary worden opgenomen.

Een databank is een model van een gebruikersmodel van de werkelijkheid.

Waarom databanken:

- Redundantie reduceren
- Data kan ‘geshared’ worden
- Gecentraliseerde controle van data
- Standaardisatie van de gegevens
- veiligheidsbeperkingen i.v.m. toegang

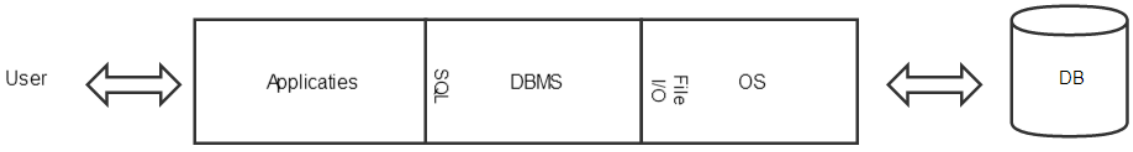
CLIENT/SERVER ARCHITECTUUR

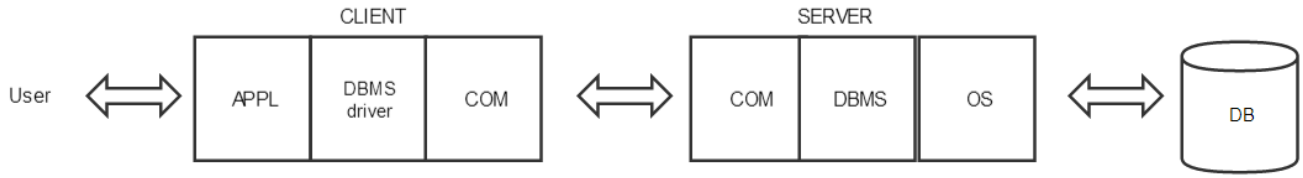
Een client/server systeem bestaat uit een netwerk van computers:

- Client-computers
  Deze bevatten applicatieprogramma's en voer applicaties uit.
  Ze bevatten DBMS-driver die de aanvragen (in SQL) voor gegevensverwerking van de applicatie ontvangt en doorgeeft aan de DBMS.
  Ze accepteren ook de antwoorden van het DBMS en zetten ze om naar de geschikte vorm voor de applicatie
- Server(s)
  Deze bevat het DBMS en de programma's voor bestandsbeheer (OS), hier wordt ook de "concurrency controle" (controle op gelijktijdige toegang tot gegevens) afgehandeld.

Zowel client-computers als servers bevatten communicatiesoftware om de berichten tussen DBMS-driver en DBMS te kunnen verzenden en ontvangen.

We maken gebruik van Pc's die voorbereidend werk kunnen leveren (ontlasten van de servers).





HET ENTITEIT-RELATIEMODEL

INLEIDING

Ontwikkelen van databasetoepassingen

- Identificeren van de entiteiten
- Beschrijven van de structuur
- Onderlinge relaties van deze entiteiten

= creëren van gebruiksgegevensmodel

Twee strategieën

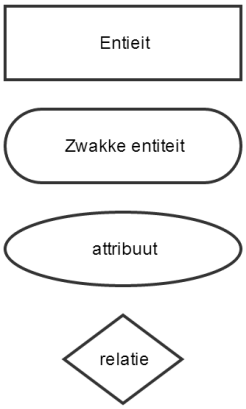
- Top-Down
  Uitwerking vanuit het gezichtspunt van de hele organisatie
  E/R is geschikter
- Bottum-Up
  Ontwikkelen specifiek systeem, om daarna verder op te bouwen
  Sneller en minder riskant

DEFINITIES

Entiteit	~tabel: onscheidbaar object uit de werkomgeving van de gebruiker
Zwakke entiteit	afhankelijk van een andere (sterke) entiteit
ID-afhankelijke entiteit	entiteit waarvan de indentifier ook de identifier van een andere entiteit bevat. Bv een appartement en zijn gebouw
Attribuut	~kolommen: beschrijven kenmerk van entiteit
Domain	Pool of legal values
Identifier	Die een entiteitsinstantie identificeren
Sleutel	Een groep van een of meer attributen die een rij in een tabel **uniek** identificeren
Relatie	Een verband tussen entiteiten, hoe hangen de tabellen aan elkaar.
Graad van de relatie	Hoeveel entiteiten in de relatie betrokken zijn, Bij E/R-model bijna alleen binaire relaties
Subtype	De eigenschappen van het supertype worden “geërfd” door het subtype.

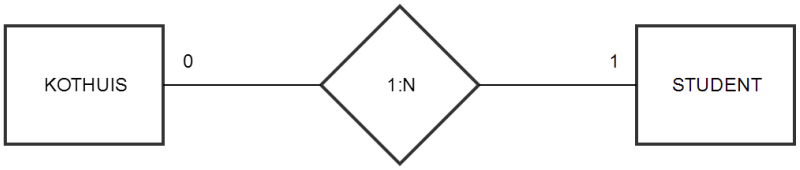
Een subtype beschrijft een IS-EEN relatie, nooit een heeft-een relatie!
Het subtype zal dezelfde sleutel hebben als het supertype.
Het supertype zal een algemeen beeld geven waarbij het subtype specifiek, specialiseert.
Bijvoorbeeld, werknemer (supertype) <-> verkoper (subtype).

E/R-DIAGRAMMEN



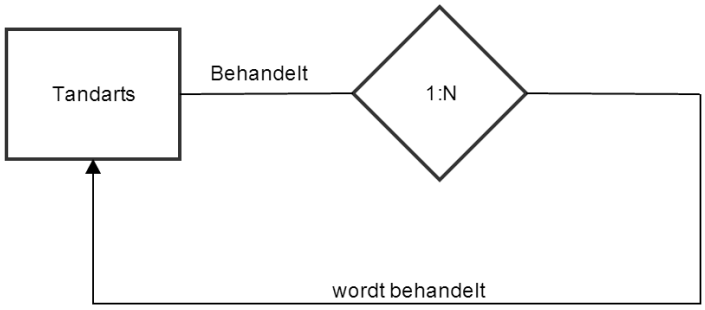
Maximale cardinaliteit
Het maximale aantal entiteiten dat met een andere entiteit kan verbonden worden.
Dit wordt weergegeven in de ruit.

Minimale cardinaliteit
Geeft het minimale verplichte aantal entiteiten weer, wat wordt weergeven op de verbinding tussen de entiteiten.



Een student moet geen kot bezitten, maar een kotbaas moet wel minstens 1 student hebben.
In één huis kunnen meerdere studenten zitten.

Recursieve relaties (heeft-een relatie)
Relaties tunnen entiteiten van dezelfde entiteitsklasse kunnen ook voorkomen.
Een werknemen kan in dezelfde werkgroep zitten met andere werknemers.



Een tandarts behandelt meerder tandartsen, maar hij wordt maar door 1 tandarts behandelt.

ALGEMENE STAPPEN OM TE KOMEN TOT EEN GEGEVENSMODEL

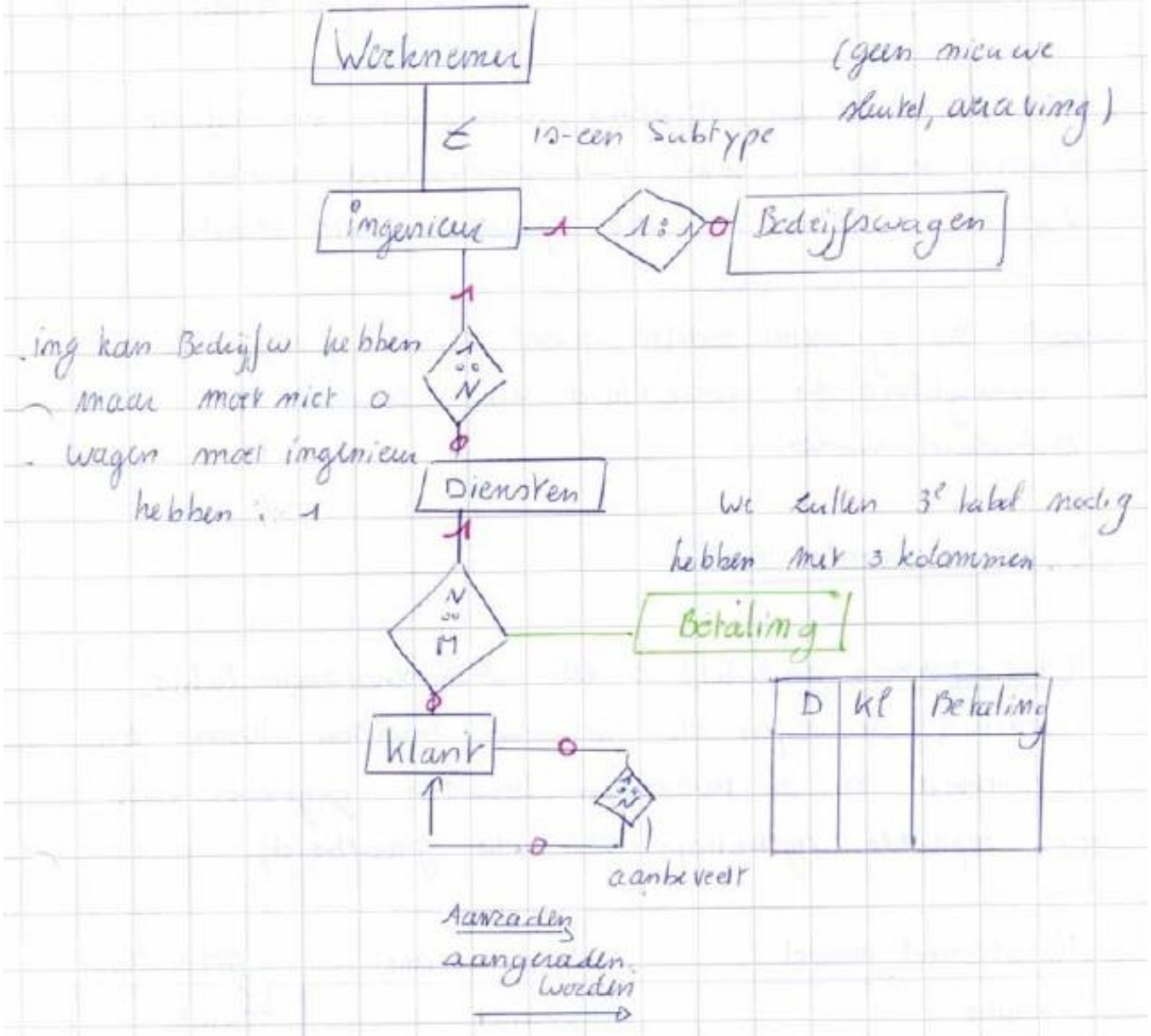
1. Identifier
   Entiteiten, attributen, identifiers, relaties, subtypen
2. Ontwerp
   een set van formele objecten
   (E/R model en E/R diagram)
3. Bedenk de formele integritietsregels
4. Bedenk de nodige formele operatoren
   (om stap 2 en 3 te implementeren)

Ontwerp procedure

- Gebruik E/R model om via top-down methodologie de “grote” relaties te ontwerpen, gebruikmakend van entiteiten,…
- Maak van deze “grote” relaties “kleinere” relaties via normalisatie

Oefening:
Een ingenieursbureau heeft een aantal medewerkers. Enkele daarvan zijn ingenieurs. Een ingenieur kan een bedrijfswagen hebben, maar iedere bedrijfswagen is toegekend aan één ingenieur. Ingenieurs verlenen diensten aan klanten. Een ingenieur kan nul of meer diensten verlenen, maar iedere dienst wordt verleend door precies 1 ingenieur. Aan een klant kunnen verschillende diensten verleend worden en dezelfde dienst aan verschillende klanten. Bij iedere klant hoort minstens 1 dienst, maar niet iedere dienst hoeft een klant te hebben. Bij deze relatie wordt de betaling bijgehouden. Som verkrijgt men een klant op aanbeveling van een klant. Aanbeveling is door hoogstens 1 andere klant mogelijk.

Oefening



Opm! is-een relatie ◇ nooit 1:N / N:M

Heeft-een relatie 1:1 / N:M / 1:N

NORMALISATIE

DEFINITIE NORMALISATIE

Normalisatie is een procedure, waardoor een relatie die bepaalde problemen bevat, kan geconverteerd worden naar 2 of meer relaties die deze problemen niet bevatten, en zo ook de werkelijkheid en correctheid van de relatie kan bevorderen.

HET RELATIONELE MODEL

Relatie:
Een 2-dimensionele tabel, waarvan de rijen de gegevens bevatten van een grootheid en de kolommen gegevens over een bepaalde eigenschap van die grootheid.

Een tabel is pas een relatie als

- Er geen dubbele rijen zijn
- Rijen ongeordend, van boven naar onder
- Kolommen ongeordend, van links naar echts
- Alle kolomwaarden zijn atomic

Of nog uitgebreider

- Iedere cel van de relatie bevat één waarde
- Iedere kolom heeft een unieke naam
- De waarden van de kolommen zijn allempaal van hetzelfde domein
- De volgorde van kolommen heeft geen belang
- **Iedere rij is uniek, er zijn geen duplicaten**
- De volgorde van de rijen is niet van belang

FUNCTIONELE AFHANKELIJKHEID

Een verzameling van attributen Y is functioneel afhankelijk van een verzameling attributen X als en enkel als iedere X-waarde met juist één Y-waarde correspondeert.

Notatie: X→Y
met X=determinant, dus als je X kent ken je Y maar niet omgekeerd (N:1 verband).

Determinant	Sleutel
Bepaald eenduidig het attribuut, maar daarom niet volledig de rij	Identificeert een rij uniek

Een sleutel is daarom altijd een determinant, maar niet omgekeerd.

Oefening 1

- Volgende tabel houdt bij welke studenten op een sportdag aan welke activiteit deelnemen en welke bijdrage ze daarvoor moeten betalen
Beperking: een student kan maar 1 activiteit doen.
Bepaal determinanten, zoek de sleutel.

StNr	St naam	Activiteit	Bijdrage
001	Jan	Schaken	2
002	Imme	voetbal	3
003	Fred	voetbal	3
004	Mathilde	Tennis	3

- Als je StNr kent, ken je dan activiteit? JA : SNr → Act
- Bepaald Snr ook de bijdrage? JA : Snr → Bijdrage
- Bepaald Activiteit de bijdrage? JA : Act → Bijdrage
- Bepaald Activiteit Snr? Nee
- Bepaald Activiteit Snaam? Nee
- Bepaald Bijdrage Activiteit? Nee
- Bepaald SNr, SNaam? JA : Snr → SNaam

Determinanten

? Is Snr Sleutel ? Is Activiteit Sleutel ?

Sleutel determineert alle andere attributen/kolommen

→ Snr is een sleutel

→ Activiteit is geen sleutel, want sport kan meerdere keren gekozen worden en Act. determineert niet Snr, Snaam!

- Sleutel kan maar 1 keer voorkomen ook of or of...
- Determinant kan meerdere keren voorkomen vb. voetbal

- De tabel is geen goede tabel, want als we vb een student schrappen vb 001 dan zijn we alle info over schaken ook kwijt.
- Als StNaam uniek is kan dat ook een sleutel zijn!

Toelichting bij "voor alle mogelijke geldige waarden in de relatie":

- Voor volgende tabel zou je tot een verkeerd besluit komen.

Leverancier	Stad	Product	Hoeveelheid
L_1	Londen	P_1	100
L_1	Londen	P_2	100
L_2	Parijs	P_1	200
L_2	Parijs	P_2	200
L_3	Parijs	P_2	300

Leverancier ⟶ Stad

Leverancier ⟶̸ Product

Leverancier ⟶̸ Hoeveelheid → Valse Afhankelijkheid, kan toeval zijn!

- Dus voor alle mogelijke geldige waarden in de relatie: Leverancier is géén sleutel want bepaald niet alle kolommen.

Leverancier & product bepalen wel de Stad en de Hoeveelheid!

(Lev, Product) ⟶ Stad

⟶ Hoeveelheid

Dus (Lev, Product) is de sleutel.

Opm! Er is altijd een sleutel in een relatie van een relationele tabel.

SLEUTELS

Sleutel	Per definitie heeft iedere relatie minstens één sleutel, dit is een set van één of meer attributen waardoor een rij in een tabel uniek geïdentificeerd wordt.
Supersleutel	De sleutel moet hierbij minimaal zijn.
Kandidaat sleutel	Als een relatie meerder sleutels heeft, dan zijn al die sleutels kandidaat sleutels en wordt er één primaire sleutel gekozen, en da anderen zijn dan de secundaire sleutels.
Vreemde sleutel	De primaire sleutel die wordt gebruikt in een andere tabel is daar de vreemde sleutel.

NORMALISATIE

Modificatie anomalieën

- Verwijderanomalie
  Wanneer bij het verwijderen van info over ene bepaald feit er ook info over een ander feit verloren gaat.
  Voorbeeld: Karel voetbalt niet meer, alle info over voetbal kwijt want hij was de enigste die voetbalde in de tabel.
- Invoeganomalie
  Als er gegevens over een bepaald feit pas kunnen ingebracht worden als er gegevens zijn over een ander feit.
  Voorbeeld: Men kan rugby niet invoegen omdat het sleutelveld niet is ingevuld.

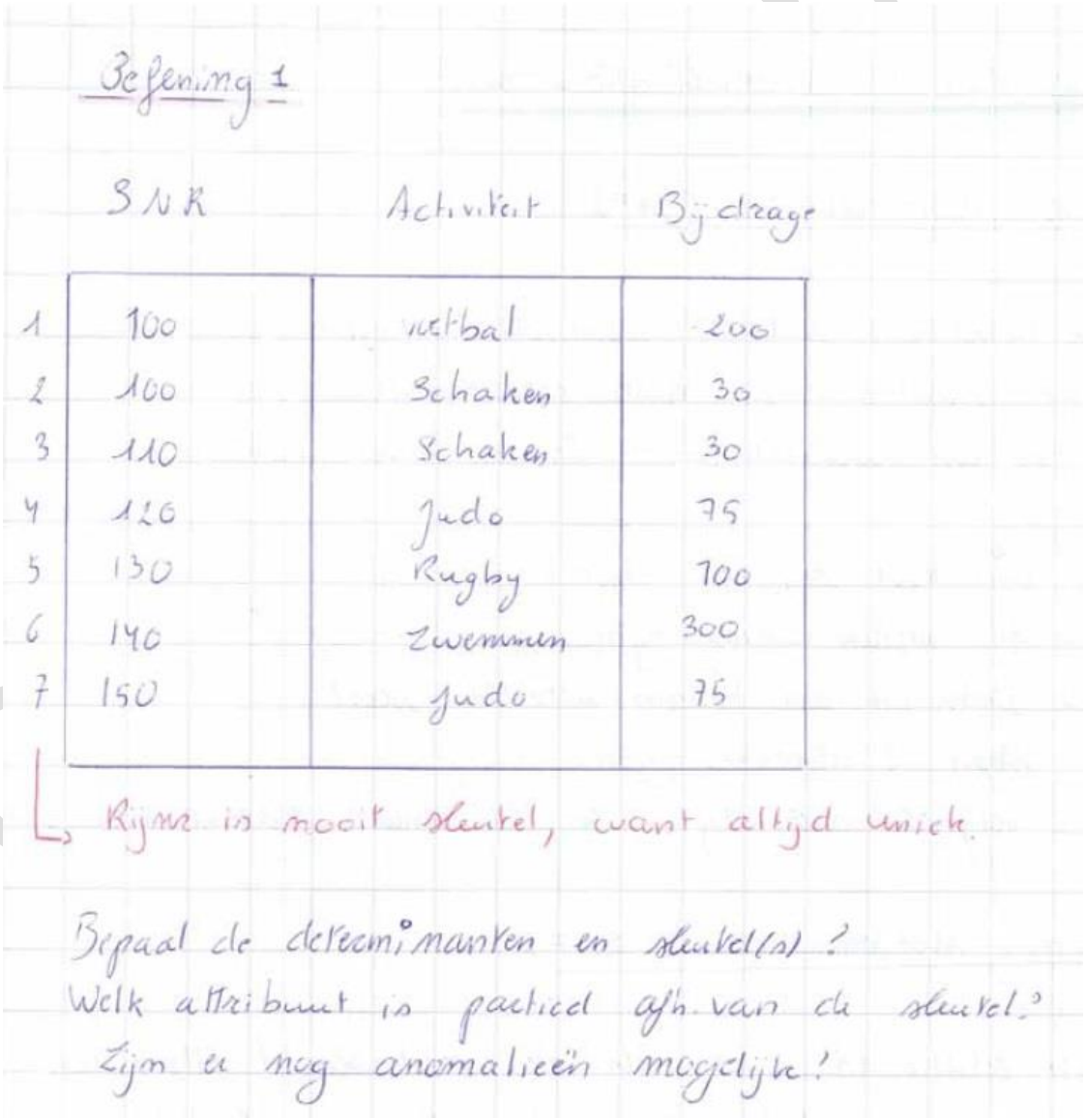
NORMALISATIE

EERSTE NORMAALVORM (1NF)

- Alle velden uniek zijn
- Data in een kolom van dezelfde soort
- Nooit 2 identieke rijen (dus zeker een sleutel)
- Volgorde rijen en kolommen maakt niet uit

TWEEDE NORMAALVORM (2NF)

Een relatie is in 2NF als alle niet-sleutel attributen functioneel afhankelijk zijn van **de volledige sleutel**, dus niet partieel afhankelijk.



Oefening 1

	SNR	Activiteit	Bijdrage
1	100	netbal	200
2	100	Schaken	30
3	110	Schaken	30
4	120	Judo	75
5	130	Rugby	700
6	140	Zwemmen	300
7	150	Judo	75

↳ Rijnr is nooit sleutel, want altijd uniek.

Bepaal de determinanten en sleutel(s)?
Welk attribuut is partieel afh. van de sleutel?
Zijn er nog anomalieën mogelijk?

funct. Afh : Act → Bijd

(Act, SNR) → Bijd → Sleutel, dus 1 NF

(Act, Bijd) ↛ SNR

Act ↛ SNR

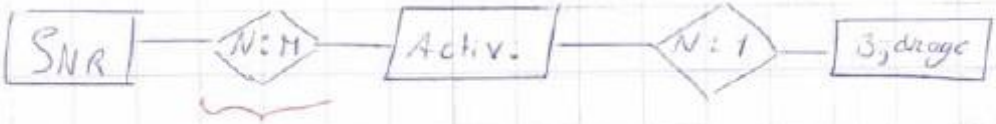
⋮

→ partiële Afh. want bijdrage hangt af van een deel van de sleutel, nl. Activiteit.

Dus Bijd, is niet Afh vd volledige Sleutel.

=> geen 2NF

goede tabel ?



- kan leiden tot een slechte tabel
- geen goede tabel want tabel heeft anomalieën

Deze tabel staat niet in 2 NF

Hoe kunnen we goede tabel maken ?

(Snr, Act) en (Act, Bijdrage)

Snr	Act
100	V
100	Z
110	Z
120	V
130	S

(Snr, Act) identiek

Act	Bijdr.
V	200
Z	100
S	50
B	50

Sleutel

- We hebben de tabel gesplitst in 2 tabellen, waarvan de Fk vd eerste, gebruikt wordt als Pk in de tweede tabel !

Oefening 2

SNR	Act	Bijdrage
2000	V	200
2100	Z	100
2200	V	200
2300	V	200
2400	V	200

Beperking : Student kan maar 1 activiteit doen.

Functionele afhankelijkheid : S → A
S → B } sleutel is hier SNR!
A → B

1NF? JA, want er is een sleutel
2NF? JA, want er is geen part. afh. elke niet-sleutel is afh van de volledige sleutel.

Toch geen goede tabel, want staat niet in 3NF!

Oefening 3

SNR	Huis	HUUR
1	BGL10	230
2	BGL10	230
3	CTSIK	210
4	WSTR	250
5	WSTR	250

SNR → Huis

Huis ↛ SNR

Huis → Huur

Huur ↛ Huis (hier wel, maar vals info) !

SNR → Huur

- SNR is de sleutel, dus 1NF

2NF = ? JA, want sleutel bestaat maar uit 1 Attribuut.

3NF = ? SNR → Huis → Huur, wel transitiviteit dus geen 3NF

DERDE NORMAALVORM (3NF)

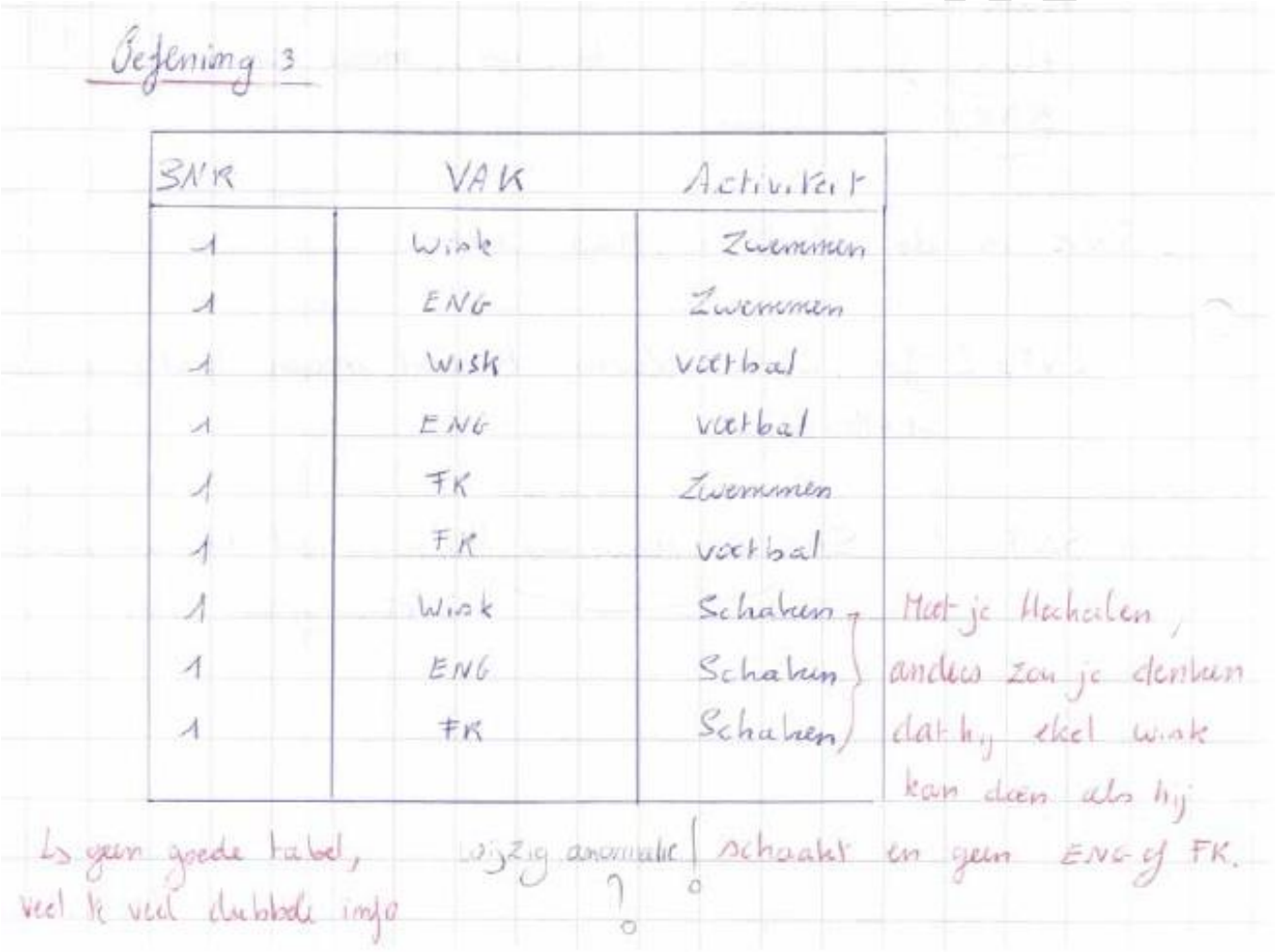
Een relatie is in de 3NF als ze in de 2NF is en GEEN transitieve afhankelijkheden bevat.

Transitieve afhankelijkheid: Als A→B en B→C dan A→C

Voorbeeld:
Een student huurt een kot in een huis. Iedereen in dat huis betaalt dezelfde huur. Als in 1 tabel bijgehouden wordt welke studenten in welk huis wonen en welke huur ze betalen, dan zondigt deze tabel tegen 3NF.

BOYCE-CODD-NORMAALVORM (BCNF)

Een relatie is in BCNF indien elke determinant een kandidaat sleutel is.



1NF? Ja want we hebben een sleutel
(SNR, Act, Vak)

2NF? Ja, want Allen zijn 1 van hun de sleutel
a/h (3 tabellen maken FK)

3NF? Ja geen transitiviteit. SNR → Vak → JAARACT
X

BCNF? Ja want elke determinant is kandidaat sleutel.

4NF? NEE

Oplossing kan zijn door ze te splitsen.

(SNR, Vak) (SNR, Act)
Sleutel (S,V) Sleutel (S,A)

↳ Ze hebben nx met elkaar te maken

VIERDE NORMAALVORM (4NF)

Een relatie is aldus in 4NF als de relatie in BCNF is en geen meerwaardige afhankelijkheid bevat.

Meerwaardige afhankelijkheid
In een relatie R(A,B,C) is er een meerwaardige afhankelijkheid als bij een waarde van A meerdere waarde van B en meerder waarden van C kunnen voorkomen, waarbij de waarden van B en C onderling onafhankelijk zijn.

Oplossing voor de anomalieën
2 relaties maken die elk data van één van de twee meerwaardige attributen bevatten.

Oefening:
Studenten (enkel hun Snr wordt in de tabel bijgehouden) leggen examens af van vakken en daarvan worden de punten bijgehouden. Van de student houdt men ook bij wat zijn/haar school was van het vorige schooljaar. In welke NF staat deze tabel.

SNR	vak	School	quotering
1	W	St-Al	14
1	E	St-Al	9
1	F	St-Al	16
2	W	H-F	15
2	E	H-F	10

1NF ?

SNR → school
A C
(SNR, vak) → quotering
(SNR, vak) → school
A B

A → C

A, B → C

Dus (SNR, vak) → Sleutel

Dus JA 1NF

2NF ?

Nee want school is slechts afhankelijk van een deel van de sleutel nl. SNR

oplossen door te splitsen in 2 tabellen !

Steven maar BCNF (elke determinant, wordt kandidaat sleutel)

(SNR, school) (SNR, vak, quotering)

PK FK

sleutel

sleutel

1

0

Appendix : Beperkingen i.v.m. normaliseren

- Het normaliseren kan soms tot veel vele verdere zaken (vb. bij queries) en dan kan denormaliseren aangewezen zijn.

Voorbeeld : Veranderd een 1:1 verband tussen gemeente en postcode.

KLANT (KL_naam, ..., Postcode, gemeente, ...)

1:1 2° tabel Stad

1:1

Stad (gemeente, postcode)

=> In niet veel info dat eruit gaat, dus niet nodig, zal systeem wel verbieden

=> We maken geen 2° tabel maken, is niet nodig!

- Er zijn soms meerdere oplossingen mogelijk, waarbij niet afgewogen worden wat de meest optimale oplossing is voor dat probleem.

Voorbeeld : Aan de universiteit in een land kan maar 1 rector zijn, maar afh. van de universiteit tot maximaal 3 vice-rectoren

. iedere opl heeft zijn modulen

opl 1 Univ (Unaam, rector, vice-rector)

X	A	P
X	A	Q
X	A	R

=> Dubbele informatie!

opl2 :

Naam } geen sleutel, bepaald niet eenduidig
Sector } een rij

2 kolommen maken

Univ

Unaam	sector
X	1

Univ1

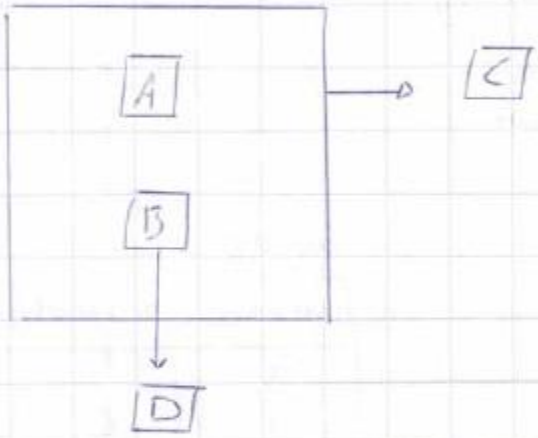
Unaam, Vicesector	
X	P
X	Q
X	R

PK → FK

=> verzaagd het Systeem.

opl3 : Univ(Unaam, sector, VK1, VK2, VK3)

Examen Oefening



A	B	C	D

- $(A, B) \rightarrow C$
- $B \rightarrow D$

- Sleutel : (A, B)
  augmentation $(A, B) \rightarrow D$
- 1NF ? JA, we hebben een sleutel
- 2NF ? Nee want D is partieel afh van de sleutel namelijk alleen van B.
- oplossing ? dat BCNF, dus iedere determinant = kandidaat sleutel
  $B \rightarrow D$ hoort niet in de tabel.
  tabel Splitsen:

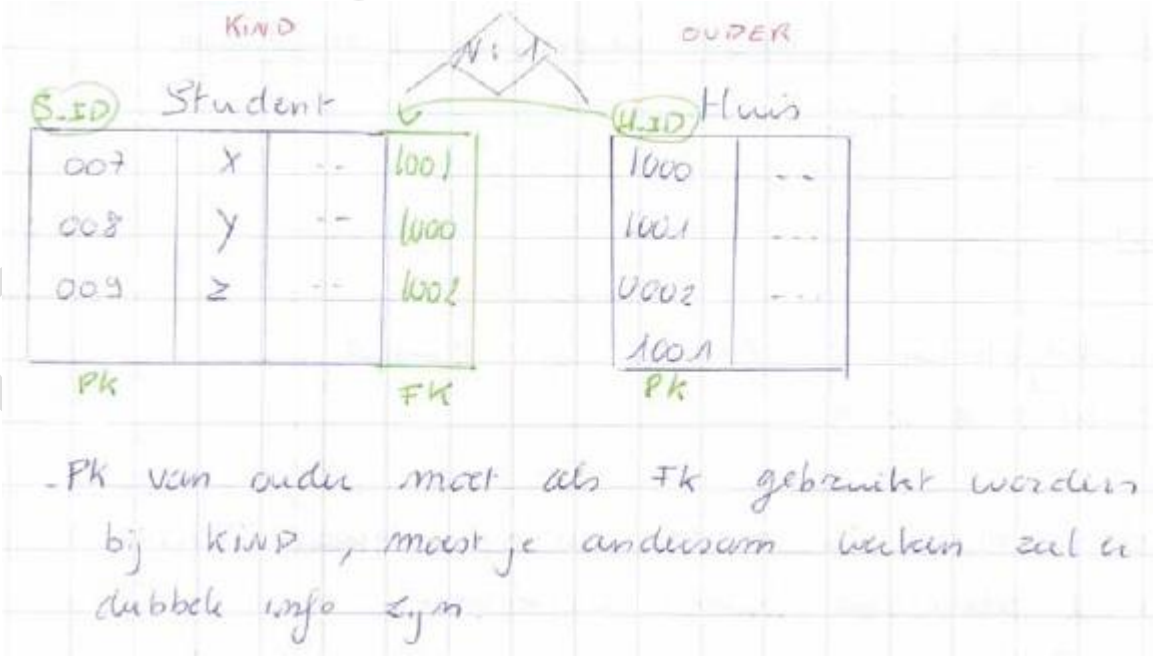
Stel tabel (A, B, C) : 1NF ? ja sleutel (A, B)	Stel (B, D)
2NF ? JA	1NF ? sleutel B
3NF ? JA	2NF ? Ja
BCNF ? JA	3NF ? Ja

ONTWERPBESLISSINGEN

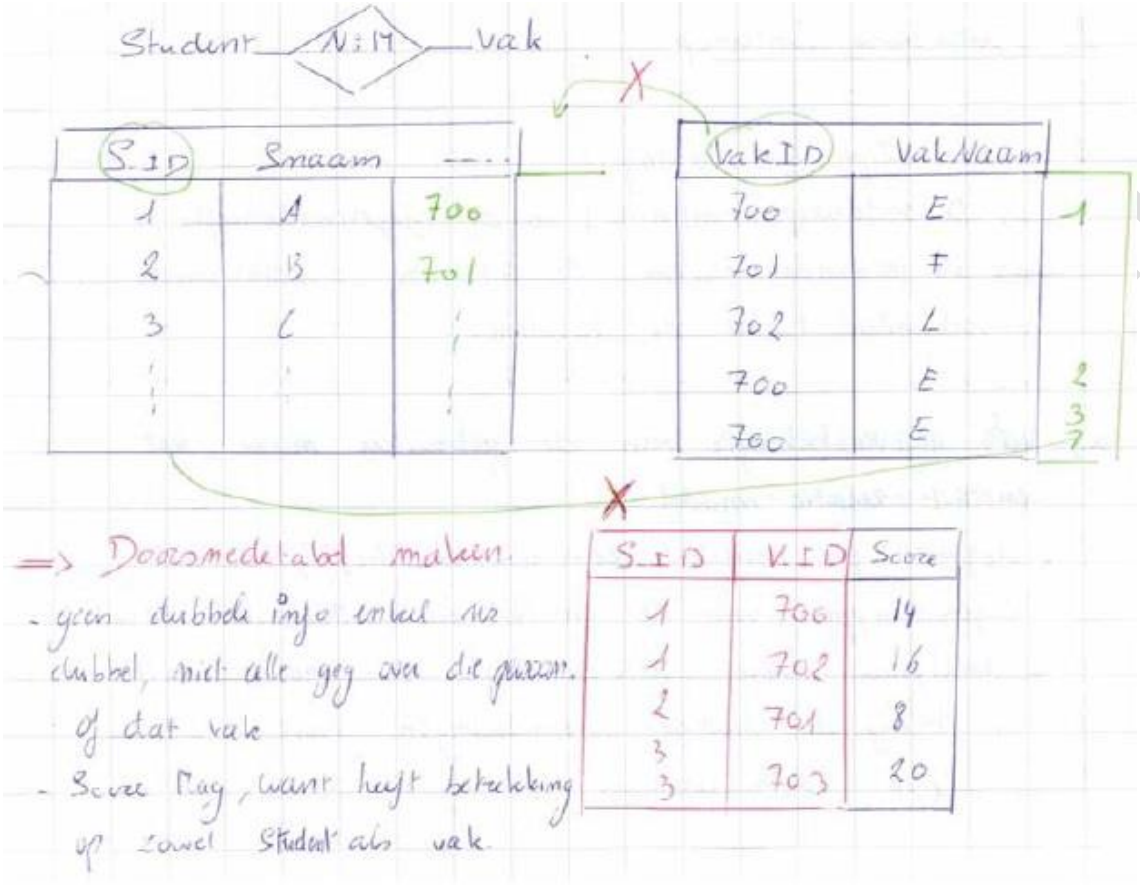
RELATIESYNTHESE

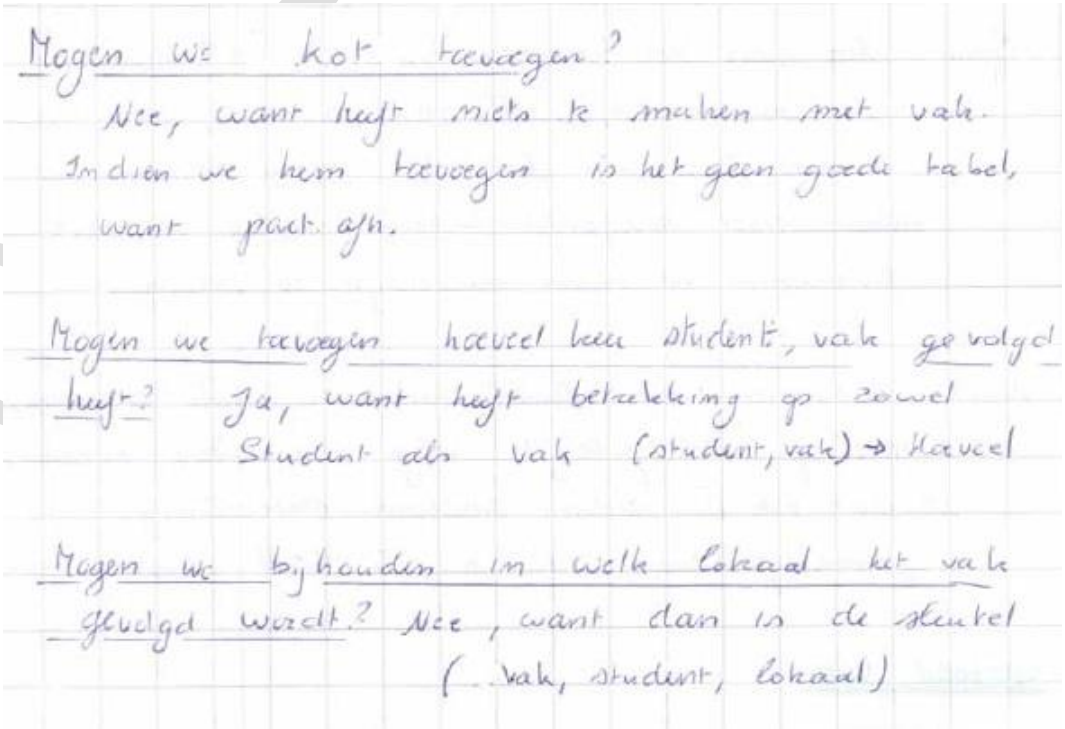
- Één-op-één relatie
  Afhankelijkheden: A→B of B→A
  Sleutel: A of B

- Veel-op-één relatie
  Afhankelijkheden: A→B
  Sleutel: A
  Attribuut C mag slechts toegevoegd worden als A→C



- Veel-op-veel relatie
  Afhankelijkheden: noch A→ B, noch B→ A
  Sleutels: (A,B), A én B dus
  Attribuut C mag slechts toegevoegd worden als (A,B)→C
  Dus als door toevoeging C, de sleutel (A,B,C) wordt dan hoort C niet thuis in die tabel.



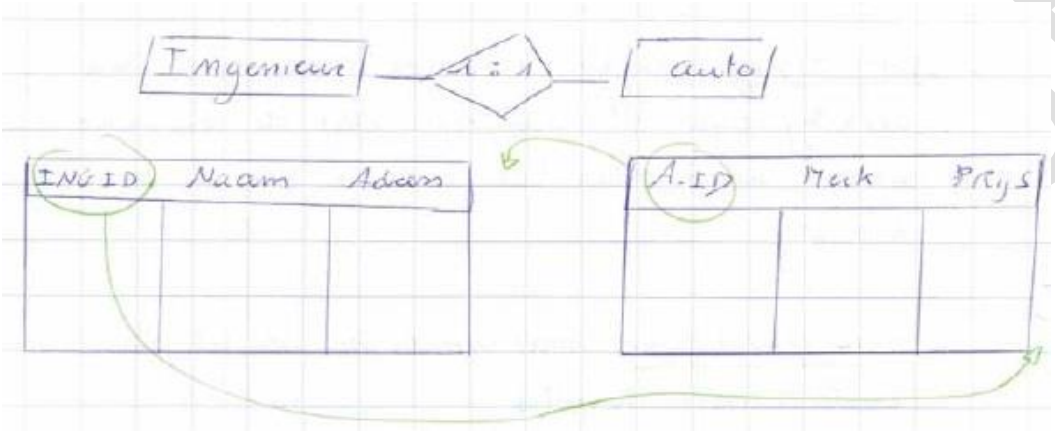


DATABASE ONTWERP

De binaire HEEFT-EEN relatie

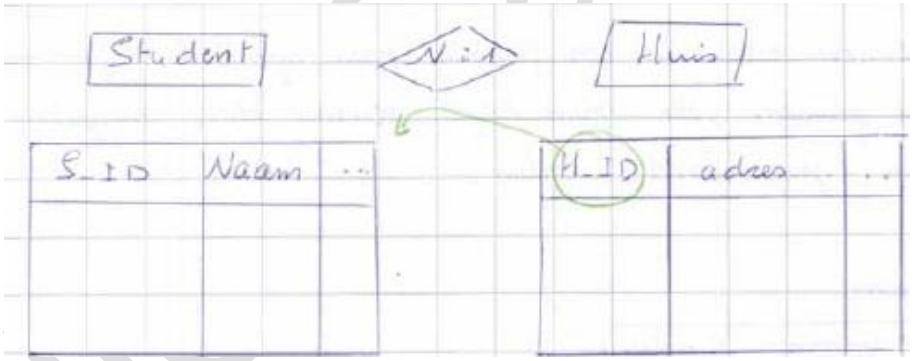
1:1

Creëer voor iedere entiteit een tabel en neem de sleutel van één van beide tabellen op in de andere tabel.
Deze sleutel wordt vreemde/ foreign key genoemd.



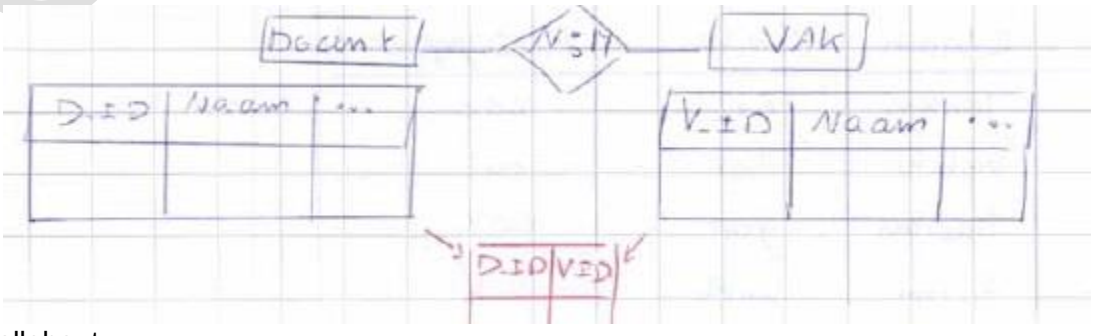
1:N

De “ouder” staat aan de 1-kant en het “kind” is de entiteit aan de N-kant.
De sleutel van de ouder wordt in de kant -tabel geplaatst. Dit moet om redundantie te vermijden.



N:M

Om redundantie en anomalieën te voorkomen, moet er een doorsnede-tabel gecreëerd worden.
Splits daarom de oorspronkelijke N:M in twee 1:N relaties, waarbij de sleutel van de doorsnede-tabel gevormd wordt door de sleutels van de 2 ouders.



RECURSIEVE RELATIES

1:1 recursieve relatie

Een persoon kan sponsor zijn van 1 andere persoon, en men kan slechts door hoogstens 1 persoon gesponsord worden.

Persoon

P_ID	P_Naam	P_adres
001	Jansens	BxL
002	Vanbeke	Antw
003	Maertens	Gent
004	Pieters	Sint-K

We hebben 1 tabel, maar we doen of ze 2x bestaat.

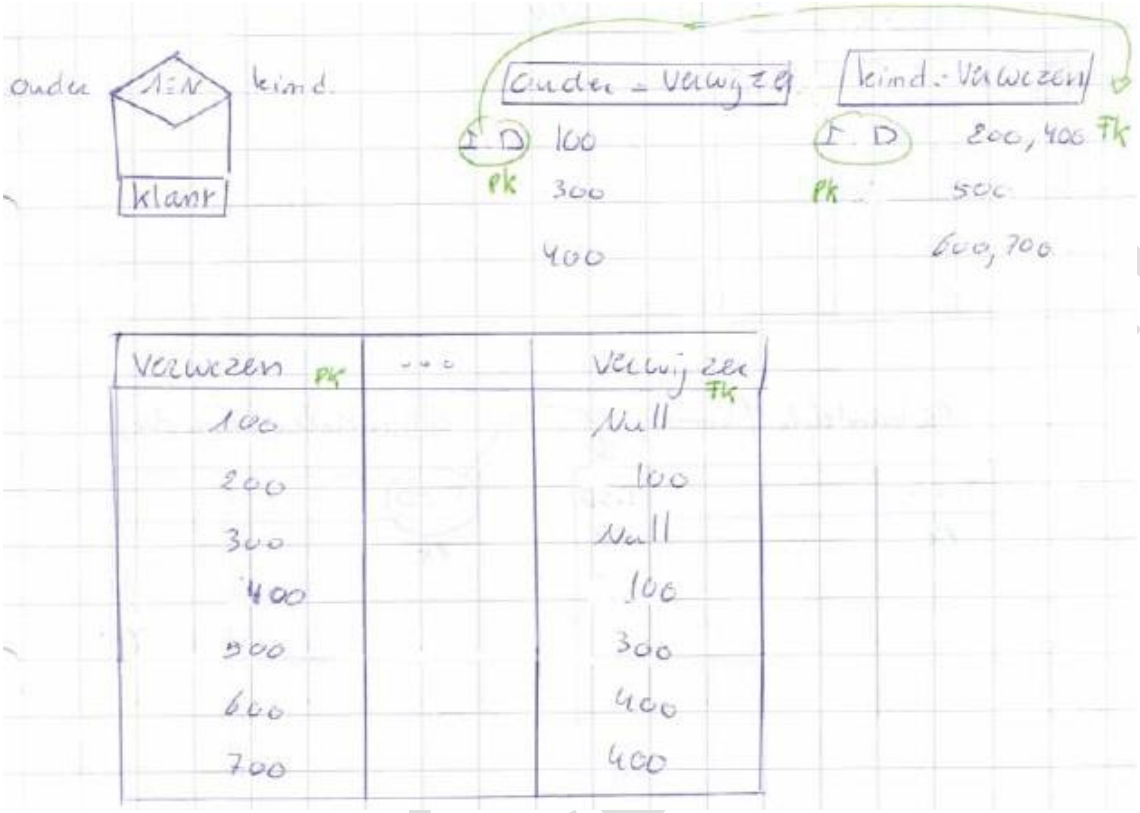
SPONSOR (ID, naam, ...)

GESPONSERD (ID, Naam)

P_ID	P_Naam	P_adres	P_ID(gesponsord)
001	Jansens	BxL	002
002	Vanbeke	Antw	Null
003	Maertens	Gent	004
004	Pieters	Sint-K	Null

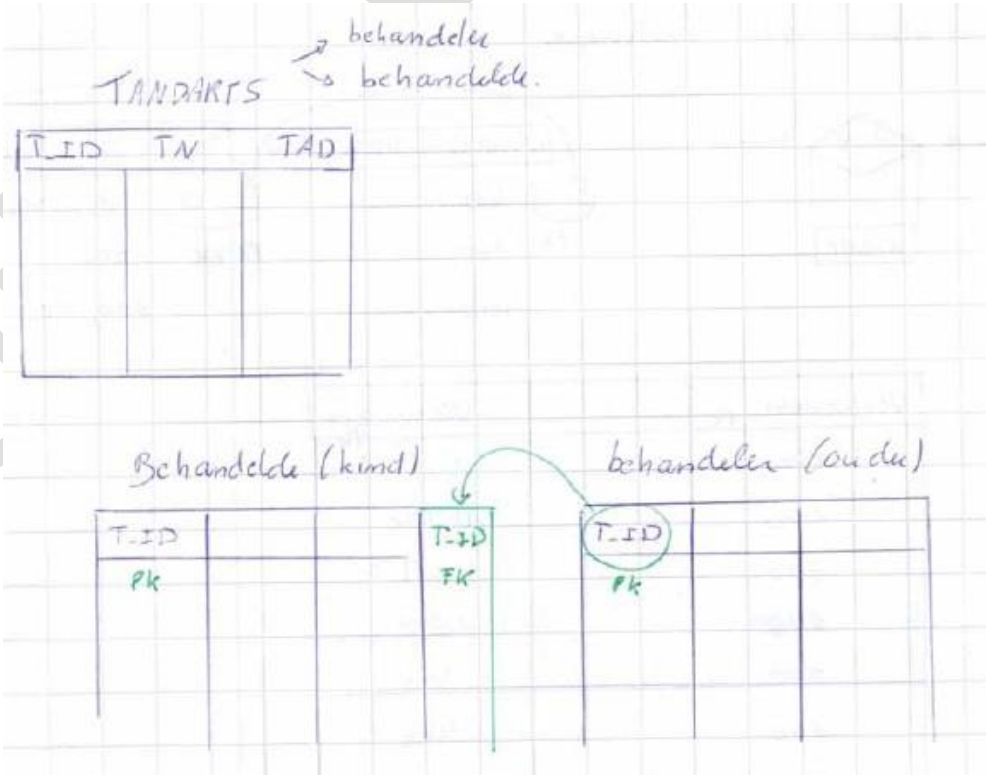
1:N recursieve relatie

Een klant kan door meerde klanten aangeraden worden om in een winkel aankopen te doen, maar iedere klant wordt finaal maar door 1 klant echt overtuigd.



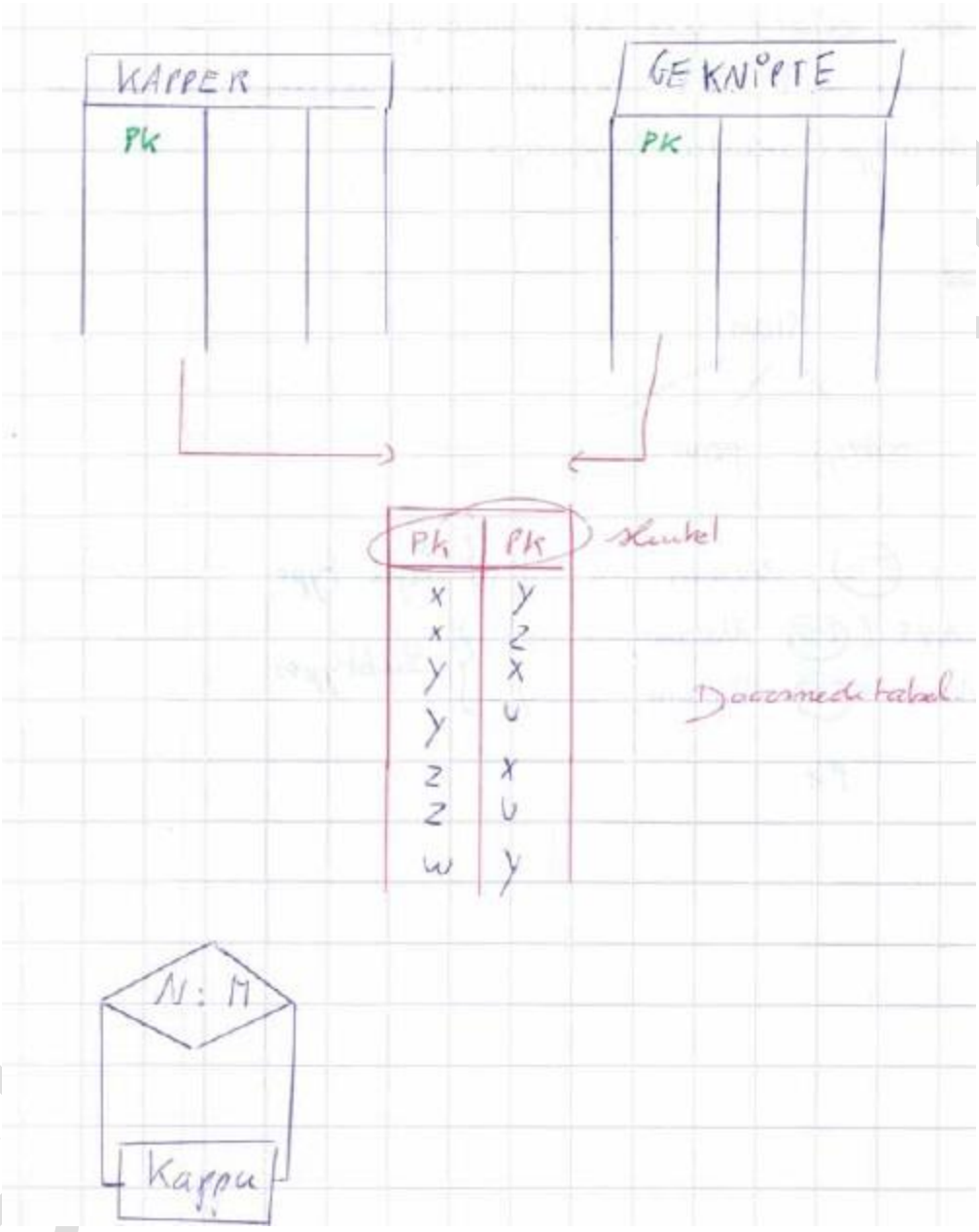
Verwezen PK	...	Verwijzer FK
100		Null
200		100
300		Null
400		100
500		300
600		400
700		400

1 Tandarts kan meerdere tandartsen behandelen, maar hij wil maar door 1 tandarts behandeld worden.



N:M recursieve relatie

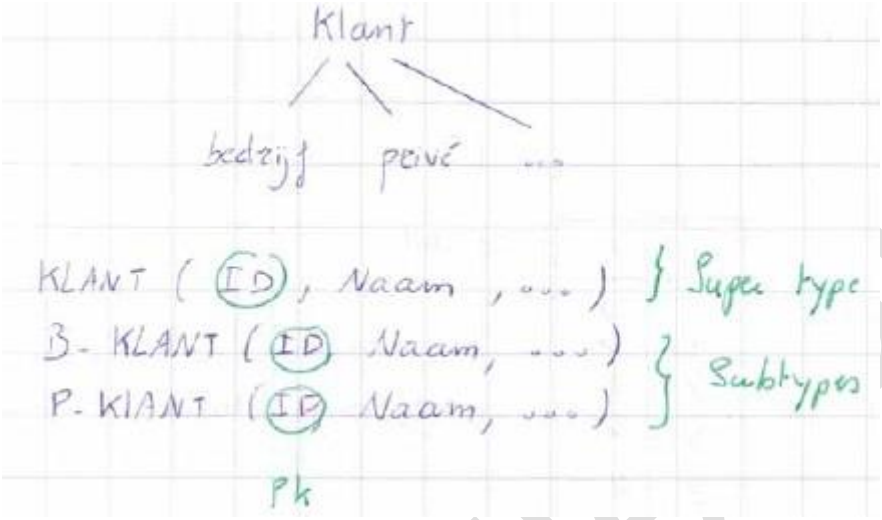
Een kapper kan het haar van meerdere kappers kappen, en laat zijn haar door meerder kappers knippen.



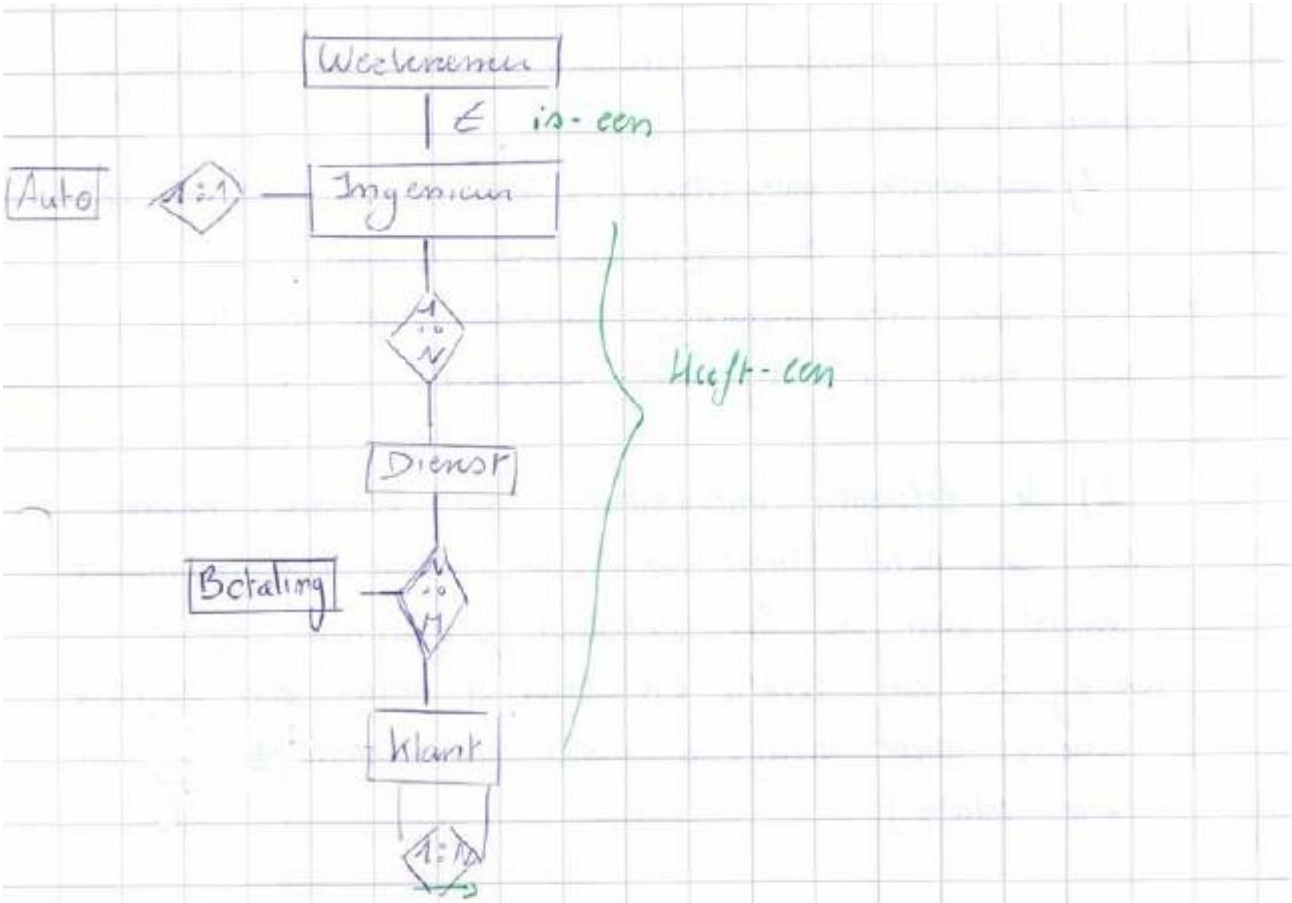
De binaire IS-EEN relatie

Deze relatie bestaat bij entiteiten die subtypen zijn binnen een bepaalde klasse.

Er wordt een relatie gecreëerd voor het supertype en een relatie voor het subtype.
Vervolgens wordt de sleutel van het supertype aan de subtype-sleutel toegevoegd.



Oefening:
Een ingenieursbureau heeft een aantal medewerkers. Enkele daarvan zijn ingenieurs. Een ingenieur kan een bedrijfswagen hebben, maar iedere bedrijfswagen is toegekend aan één ingenieur. Ingenieurs verlenen diensten aan klanten. Een ingenieur kan nul of meer diensten verlenen, maar iedere dienst wordt verleend door precies 1 ingenieur. Aan een klant kunnen verschillende diensten verleend worden en dezelfde dienst aan verschillende klanten. Bij iedere klant hoort minstens 1 dienst, maar niet iedere dienst hoeft een klant te hebben. Bij deze relatie wordt de betaling bijgehouden. Som verkrijgt men een klant op aanbeveling van een klant. Aanbeveling is door hoogstens 1 andere klant mogelijk.



Werknemer (W_ID, Naam, ...)
PK

Ingenieur (W_ID , Naam, ...)
PK

Auto (A_ID, ... , W_ID)
PK FK

Dienst (D_ID, ... , W_ID)
PK FK

Klant (KL_ID, KL_Naam, KL_ID)
PK FK
aanzaden als FK in aangeraden tabel.

KL-D (D-ID, KL_ID , Betaling)
Dienst-aan-klant tabel. sleutel.

INTEGRITEIT

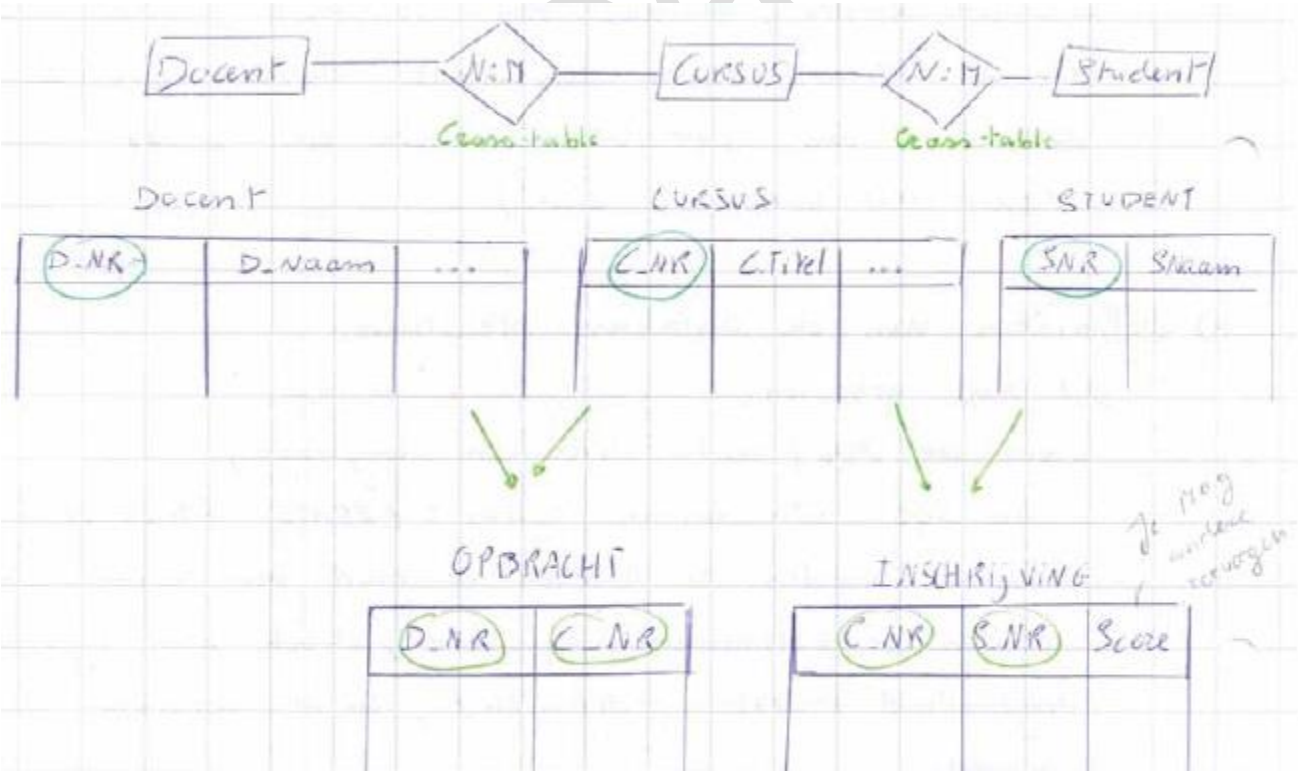
Het relationele model is frequent beschreven, inclusief 2 integriteitsregels:

1. De entiteit integriteit (PK)
   Geen enkele primaire sleutel kan een NULL-waarde hebben
   (er kan geen info opgeslagen worden over iets dat niet kan geïdentificeerd worden)
2. De referentie integriteit (FK)
   Een vreemde sleutel van een tabel moet verwijzen naar een primaire sleutel van de in verband gebrachte rij.
   (een rij in één relatie die verwijst naar een andere relatie moet verwijzen naar een bestaande rij in de relatie)
3. De attribuut integriteit
   Ieder attribuut moet voldoen aan de beperking dat zijn waarde komt uit een relevant domein.
   (*voorbeeld: Geboortejaar kan niet bestaan uit 21bc*)

DATABASE IMPLEMENTATIE

Definiëren van de database-structuur

Voorbeeld:
Een docent geeft meerdere cursussen, die elk gevolgd worden door meerdere studenten.



```
CREATE TABLE DOCENT          niet je invullen, want is PK
( D_NR  INTEGER  NOT NULL  PRIMARY KEY,
  D_NAAM  VARCHAR(40)  NOT NULL ,
  D_ADRES  VARCHAR(80)  ,
)
```

```
CREATE TABLE CURSUS
(C_NR  Integer (ook: Serial)  NOT NULL  PRIMARY KEY,
 C_Titel  VARCHAR(80)  NOT NULL,
)
```

Naam mag je kiezen, maar later wel koppeling maken.

```
CREATE TABLE OPDRACHT
( Cursus_Nr  Integer  NOT NULL,
  Docent_Nr  Integer  NOT NULL,
  CONSTRAINT  Opdracht_PK  PRIMARY KEY (Cursus_NR,
                                        docent_NR)
  CONSTRAINT  Opdracht_FK1  FOREIGN KEY (docent_NR),
              REFERENCES  DOCENT (D_NR),
  CONSTRAINT  Opdracht_FK2  FOREIGN KEY (cursus_NR),
              REFERENCES  CURSUS (C_NR)
)
```

Basisbegrippen van relationele algebra

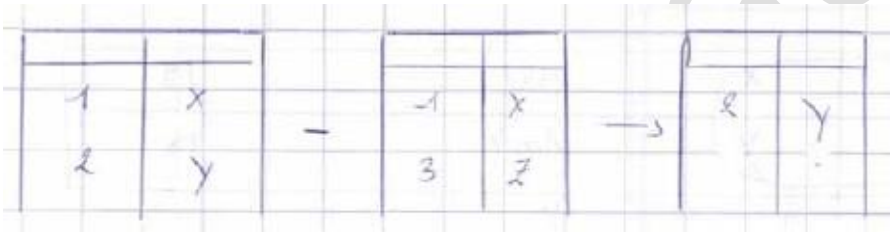
De bewerking op één of meer relaties is altijd opnieuw een relatie, daarom noemt men de relationele algebra: “gesloten”.

De unie

De rijen van de eerste relatie worden toegevoegd aan alle rijen van de tweede tabel toe te voegen.

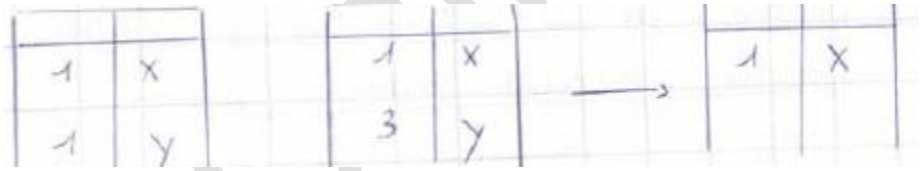
Het verschil

De derde tabel bevat die rijen van de eerste die niet in de tweede voorkomen.



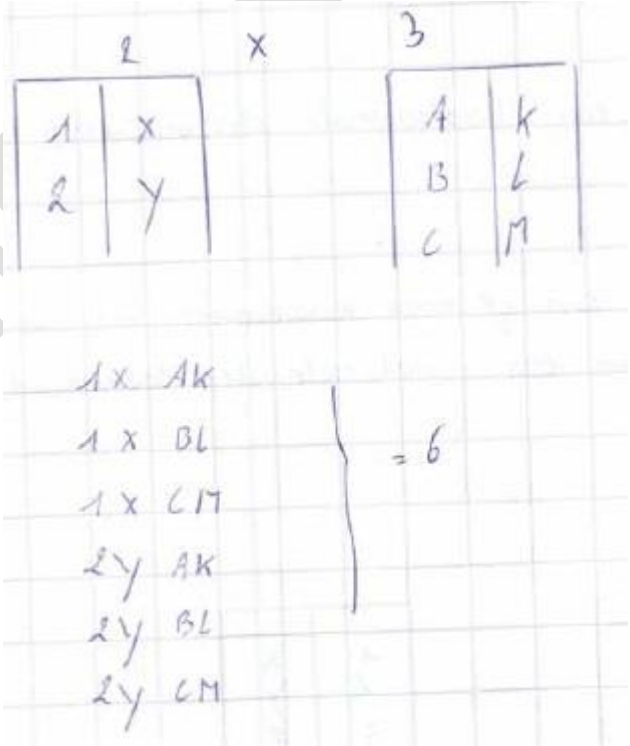
De doorsnede

De derde relatie bevat die rijen, die zowel in de eerste als in de tweede voorkomt.

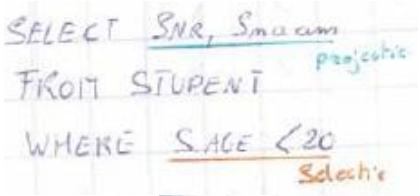


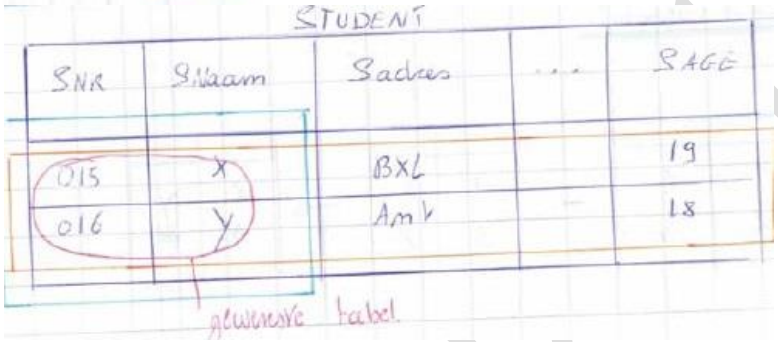
Het cartesisch product

Alle mogelijke combinaties maken met de rijen.



De selectie	Maakt een horizontale deelverzameling uit een tabel
De projectie	Selecteert één of meer kolommen uit een tabel het maakt dus een verticale deelverzameling uit een tabel

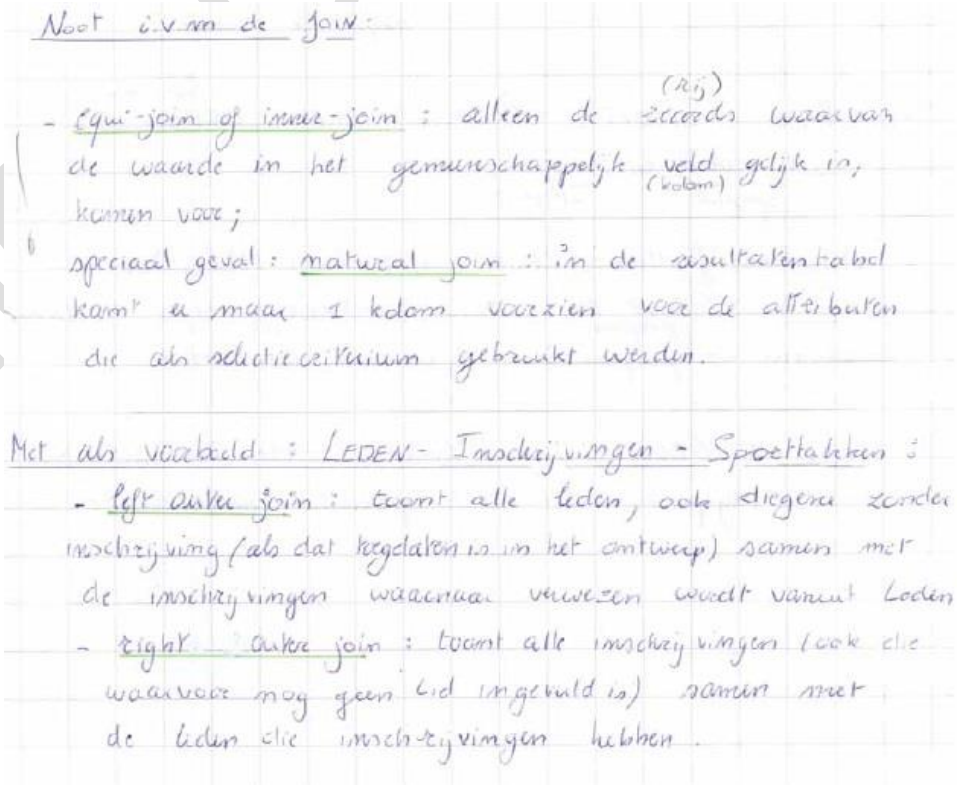




Join

Voegt rijen samen uit 2 relaties op grond van gespecifieerde attribuutwaarden
het is het resultaat van 3 bewerkingen:

1. A*B
2. Selectie volgens criterium
3. Projectie van de aangegeven kolommen
   Hier wordt altijd een natural join bedoeld
   de attributen in de voorwaarde moeten ook een gemeenschappelijk domein hebben


Noot i.v.m de join:

- equi-join of inner-join: alleen de (rij) records waarvan de waarde in het gemeenschappelijk veld (kolom) gelijk is, komen voor;
  speciaal geval: natural join: in de resultatentabel komt er maar 1 kolom voorzien voor de attributen die als selectiecriterium gebruikt werden.

Met als voorbeeld: LEDEN - Inschrijvingen - Sporttakken:

- left outer join: toont alle leden, ook diegene zonder inschrijving (als dat tegelaten is in het ontwerp) samen met de inschrijvingen waarnaar verwezen wordt vanuit Leden
- right outer join: toont alle inschrijvingen (ook die waarvoor nog geen Lid ingevuld is) samen met de leden die inschrijvingen hebben.

QUERIES UIT 1 TABEL

```
SELECT kolommen --projectie
FROM tabel
WHERE selectievoorwaarden --selectie
ORDER BY kolom ASC | DESC --sorteren
```
(meer in Hoofdstuk hierna)

QUERIES UIT MEERDERE TABBELEN

Met subqueries

```
SELECT kolomm
FROM tabel1
WHERE selectievoorwaarden
IN (
      SELECT kolomm
      FROM tabel2
      WHERE selectievoorwaarde
      )
```
Deze werkwijze werkt maar goed zolang de attributen van het resultaat uit slechts 1 tabel komen, indien dit niet het geval is, moeten tabellen worden samengevoegd (join).

Alsook moet het resultaat van de tweede SELECT overeenkomen met attribuut dat in de WHERE werd gebruikt.

Voorbeeld:
Wat zijn de namen van de studenten die zich ingeschreven hebben voor Engels?

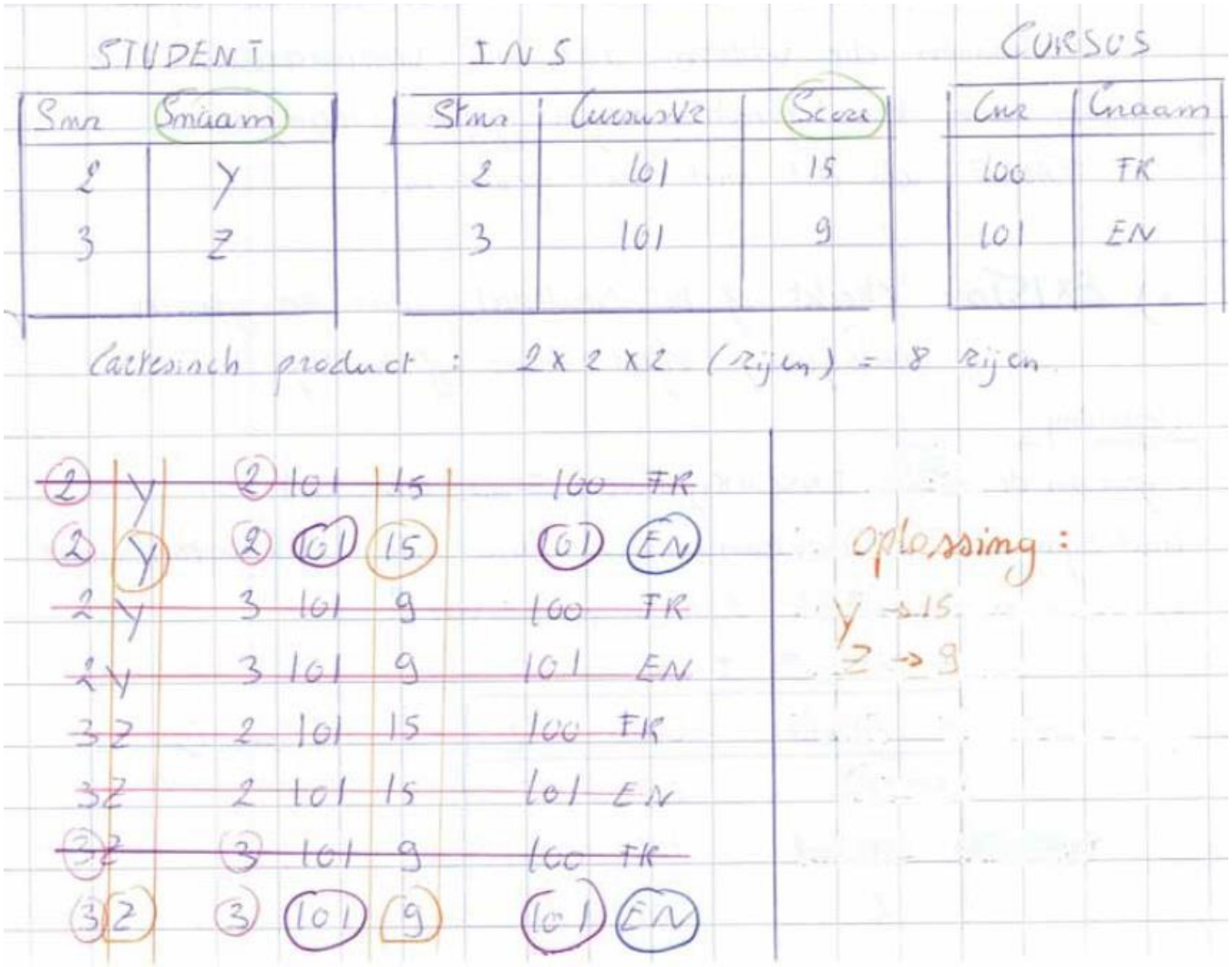
VIA SUBQUERY

```
SELECT Snaam -- resultaat = Y en Z
FROM STUDENT
WHERE Snr
IN ( -- resultaat = 2 en 3
      SELECT Stnr
      FROM INS
      WHERE Cursusnr
      IN ( -- resultaat = 101
            SELECT Cnr
            FROM CURSUS
            WHERE Cnaam='EN'
            )
      )
```

Zwakke punt van Subquery is dat het efficiënt is zolang dat de oplossing maar uit 1 tabel moet komen, dus redelijk beperkt.

VIA JOIN

Geef de naam van de studenten weer die Engels volgen met hun score



1. A*B
2. Selectie volgens criterium
3. Projectie

```
SELECT STUDENT.Snaam, INS.score --projectie
FROM STUDENT, INS, CURSUS -- cartesisch product
WHERE STUDENT.Snr = INS.St.nr
      AND INSCH.Cursusnr = CURSUS.Cnr
      AND Cursus.Cnaam = 'EN' -- selectie (3 VW)
```

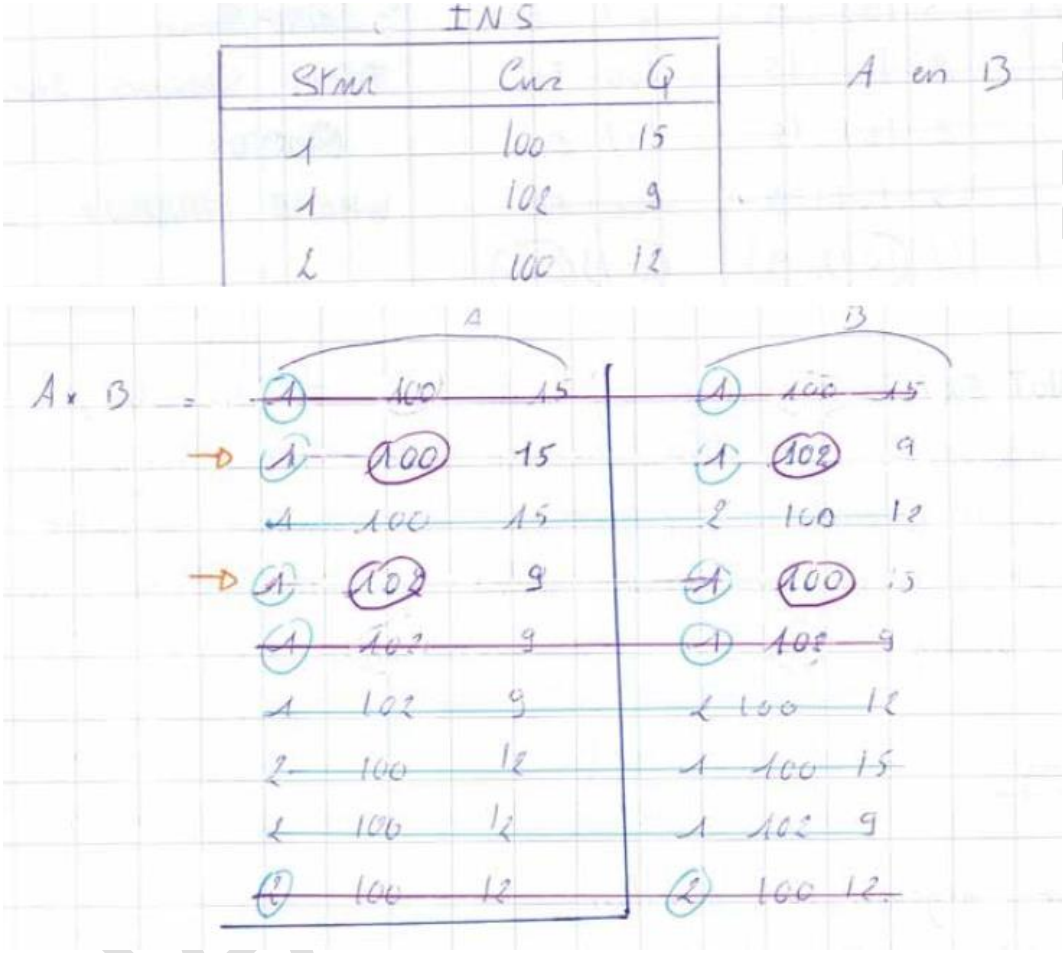
EXIST EN NOT EXIST (SUBQUERIES)

Zijn logische operatoren, hun waarde is dus TRUE als er rijen voorkomen die voldoen aan de voorwaarden die in hun kwalificaties zijn geformuleerd (omgekeerd FALSE).

EXIST

Voorbeeld:

Wat zijn de studentenummers van de studenten die op meer dan 1 vak zijn ingeschreven?



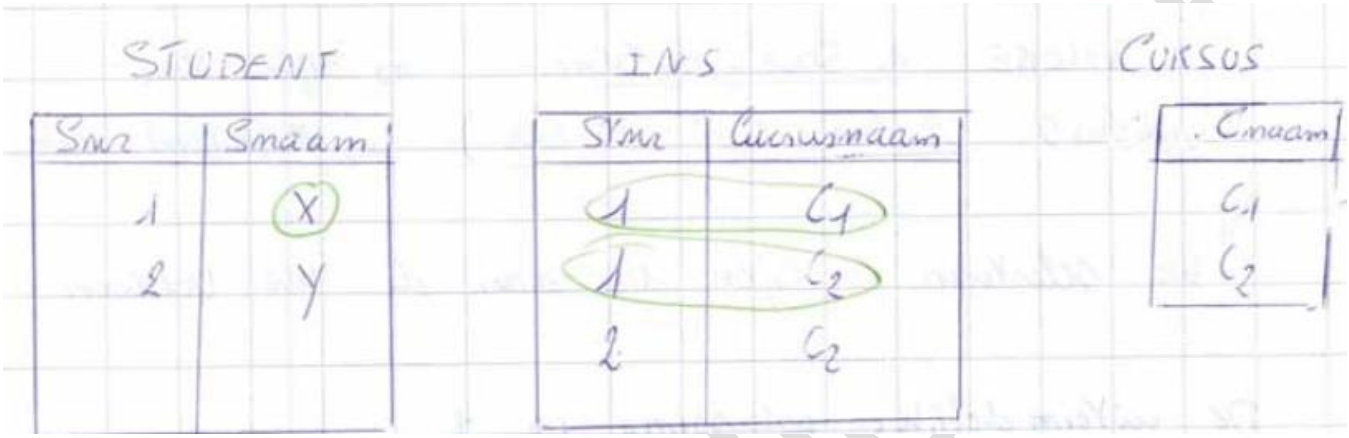

```
SELECT DISTINCT Stnr
FROM INS AS A -- enkel A deel wordt geselecteerd
WHERE EXISTS ( -- TRUE want subquery bevat rijen
      SELECT *
      FROM INS AS B
      WHERE A.stnr =B.stnr -- zelfde Stnr
      AND A.Cnr NOT = B.Cnr -- verschillend vak
      )
```

NOT EXIST

Iedere rij van de FROM-tabel waarvoor het resultaat van de query leeg is wordt geselecteerd.

Voorbeeld:
Wat zijn de namen van de studenten die zich op ALLE cursussen hebben ingeschreven?

Vraag veranderen:
Zijn er studenten waarvoor NIET geldt dat er vakken zijn waarvoor ze zich NIET hebben ingeschreven.




```
SELECT Snaam
FROM STUDENTEN
WHERE NOT EXISTS ( -- studenten waarvoor niet geldt dat er vakken zijn die...
      SELECT *
      FROM CURSUS
      WHERE NOT EXISTS ( -- ...zich niet hebben ingeschreven
            SELECT *
            FROM INS
            WHERE INS.Snr = STUDEN.Snr
            AND Cursus.Cnaam= INS.Cursusnaam
      )
)
```

Het is niet echt een cartesisch product, maar we leggen het wel uit met een cart. product.

STUDENT × INS × CURSUS

STUDENT × CURSUS

We schrappen deze rij want Student 2 is niet ingeschreven voor C_1

STUDENT

We schrappen Stud die zich niet voor Alle vakken heeft ingeschreven

opl.

STURCTURED QUERY LANGUAGE

SYNTAX	Extra uitleg
SELECT kolom_naam **FROM** tafel_naam; of SELECT * FROM tafel_naam;	Selecteren van data uit een database Selecteren van alle kolommen
SELECT **DISTINCT** kolom_naam FROM tafel_naam;	Enkel verschillende waarden selecteren
SELECT kolom_naam FROM tafel_naam **WHERE** kolom_naam operator waarde;	Selecteer enkel records die voldoen aan het criterium
SELECT kolom_naam FROM tafel_naam WHERE kolom_naam operator waarde **AND** (…**OR**...);	
SELECT kolom_naam FROM tafel_naam **ORDER BY** kolom_naam,kolom_naam ASC|DESC; **INSERT INTO** tafel_naam (kolom1,kolom2,…) **VALUES** (waarde1, waarde2,…);	 Sorteren op stijgende of dalende volgorde (default=ASC) Voeg een nieuwe rij toe met deze waardes
UPDATE tafel_naam **SET** kolom1=waarde1, kolom2=waarde2,… **WHERE** kolom=waarde;	Verander de waarden in kolom1 en kolom2 … op de plaats met kolom=waarde
DELETE FROM tafel_naam **WHERE** kolom=waarde;	Verwijder de rij waar kolom=waarde
DELETE FROM * FROM tafel_naam;	Alle rijen verwijderen ‘*’ moet er niet staan
SELECT COUNT(*) FROM tabel	Geeft het aantal geselecteerde rijen weer

SELECT COUNT**(kolom)**
FROM table

Geeft het aantal rijen weer zonder NULL waarde in de specifieke kolom

SELECT COUNT**(DISCTINCT** kolom**)**
FROM table

SELECT
AVG()
MAX()
MIN()
SUM()
FROM tabel

SELECT kolom, SUM()
FROM tafel
GROUP BY kolom

Die SUM() is een voorbeeld

Groeperen op die kolom

SELECT kolom, SUM()
FROM tafel
GROUP BY kolom
HAVING SUM(kolom) conditie

Kan ook een andere functie zijn bv. AVG()

Aliassen:
SELECT kolom **AS** alias_naam
FROM tafel;

SELECT kolom
FROM tafel **AS** alias_naam;

SELECT kolom
FROM tafel
WHERE kolom **IN[…,…]**

WHERE kolom **NOT IN […,…]**

Meerdere waarden in een WHERE selecteren