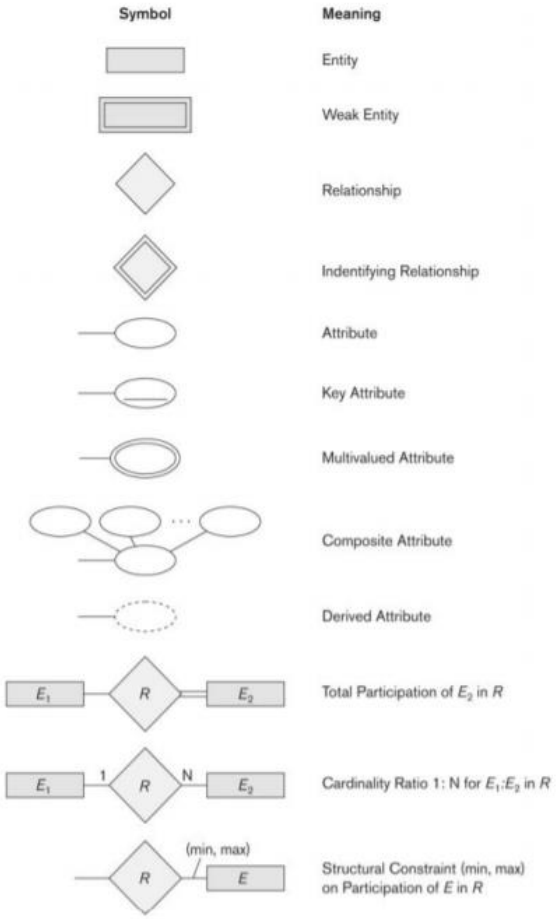
SAMENVATTING LES 2

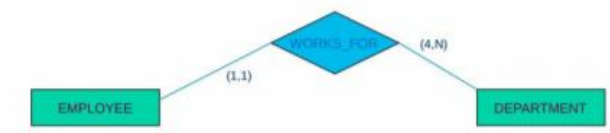
ER & EER

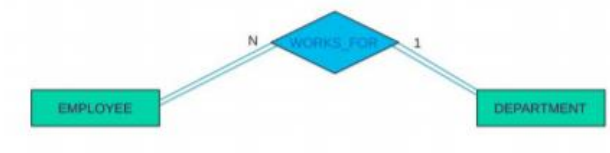
Cheat sheet

ER

1. Identify entity and their attributes
2. Identify relations between entities
3. Add cardinality; identify weak entities

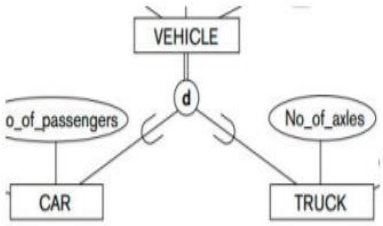


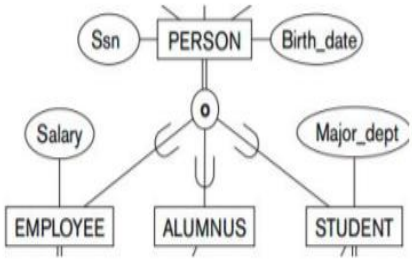


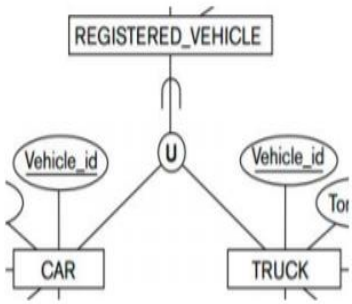


Extended ER

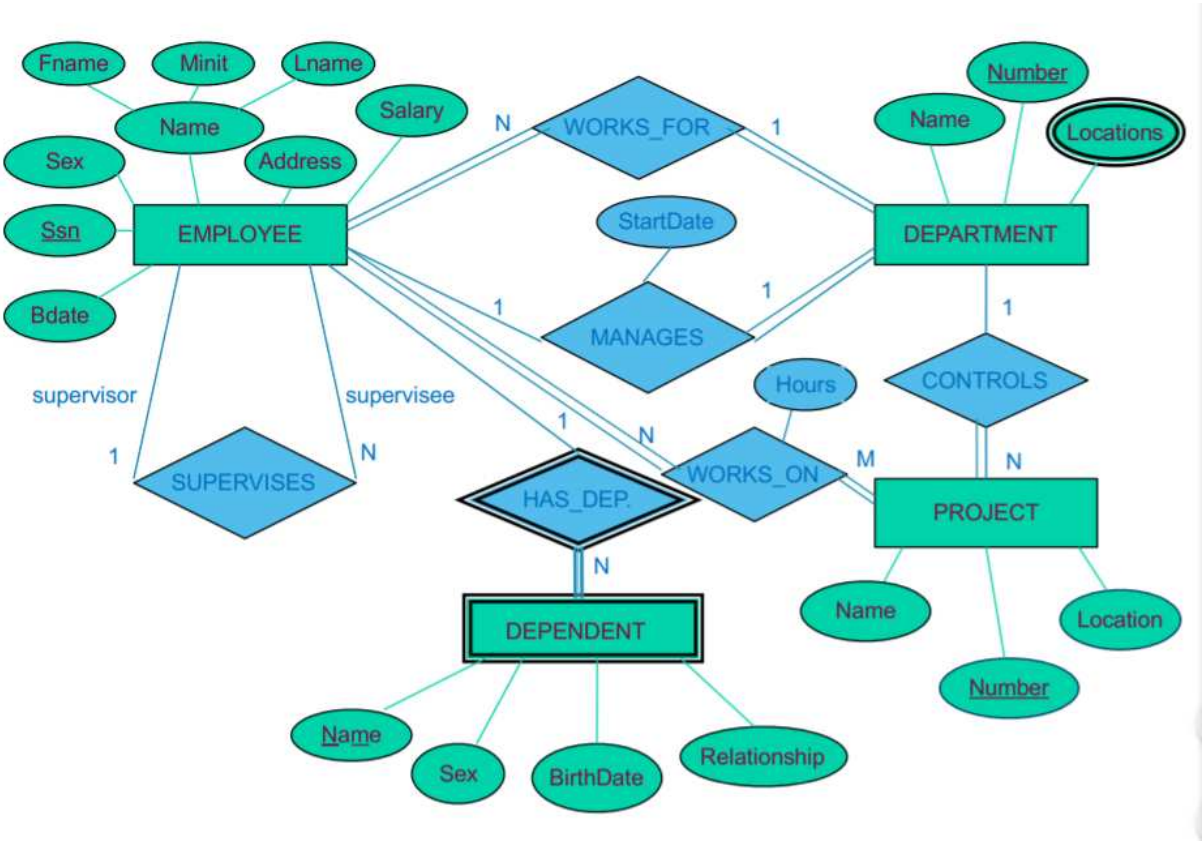
d: disjoint specialisation / generalisation
o: overlapping specialisation / generalisation
U: union







Entiteit-Relatie (ER) model



1. Ontwerp van een gegevensbank

ER & EER behoren tot het conceptuele model, het meest abstracte niveau. Op het conceptuele niveau kan je beslissingen maken onafhankelijk van welk databasesysteem.

2. Conceptueel gegevensmodel, ER model, ER diagrammen

Relevant deel van de werkelijkheid beschrijven; objecten + feiten door een schema te modelleren waarbij details weggelaten worden. Waarom? Conceptueel model staat dichter bij realiteit dan een implementatiemodel & de structuur is onbelangrijk voor een gebruiker.

Stap 1: Identificeer entiteitstypes, attributen & sleutelattributen van elk entiteitstypen

Schema = miniwereld

Entiteiten: Objecten, Acteurs (entiteittype: klasse)

Relaties: Verbanden tussen objecten, de graad is het aantal betrokken entiteitstypes. (binair, ternair, …) Meermaals 1 entiteitstype (speelt dus een rol in de relatie) → recursief relatietype

Attributen: Eigenschappen → Samengesteld/enkelvoudig, eenwaardig/meerwaardig, expliciet/afgeleid, domein (= mogelijke waarden), null-waarden (onbekend of nvt), nesten van samengestelde/meerwaardige attributen. Kunnen ook bij een relatie staan.

Sleutel = identificeert een entiteitstype/relatietype en de waarde ervan bepaalt eenduidig de entiteit/relatie. De waarden van dit attribuut verschillen voor elke verschillende entiteit.

Stap 2: Identificeer relatietypes en attributen ervan (connectiviteit & deelnamebeperkinge) + evt afgeleide attributen in entiteitstypes

Beperkingen:

- Connectiviteit: binair → 1:1 (attribuut bij een van de entiteiten)
  1:n (attribuut aan n-zijde)
  m:n (attribuut moet bij relatietype)
  ternaire, … → meer mogelijkheden
- Deelnamebeperking: totaal/partiele deelname, totaal is vaak bestaansafhankelijk
  Binair: T-T, T-P of P-P

Zwakke entiteitstypes: Totale deelname in identificerende relatie, exact 1 eigenaar, partiële sleutel. Deze sleutel (= 'discriminator') identificeert entiteit binnen entiteiten met eenzelfde eigenaar. Zwakke entiteitstypes enkel gebruiken indien nodig, moet minstens identiteisafh zijn en soms bestaansafh

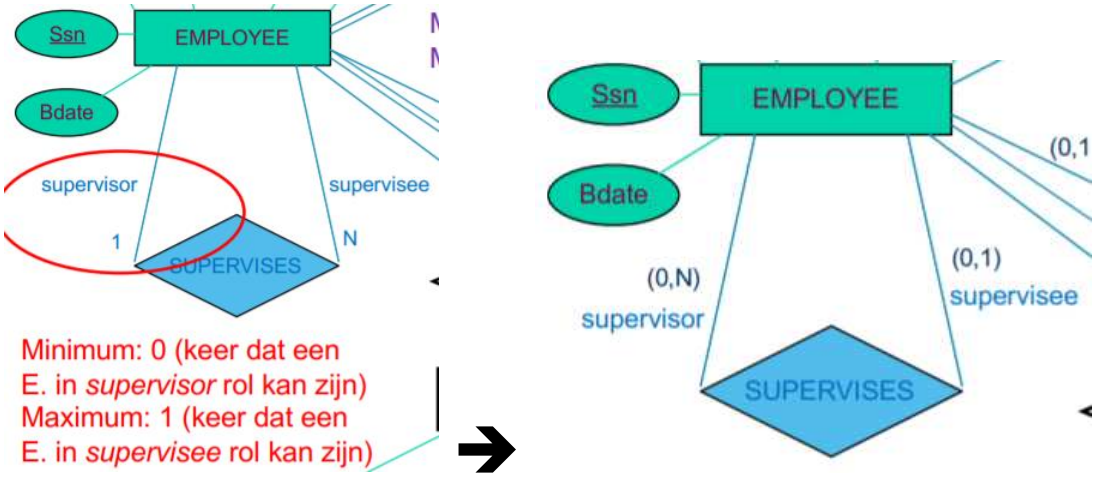
- Bestaansafhankelijkheid: bestaan ene hangt af van bestaan andere (ondergeschikt - dominant entiteit)
- Identificatieafhankelijkheid: eigen attributen identificeren niet (→ zwakke entiteit)

Alternatieve notatie voor de connectiviteit: (min, max) = hoe vaak entiteit rol speelt i/e relatie

Min = 0 → Partieel, min > 0 → totaal

Expressiviteit:
- binaire relatietypes & min ∈{0,1} & max ∈{1,n}: equivalent
- Min, max kunnen andere integers zijn → expressiever
- Graad > 2: combinatie van de 2 notaties is expressiever

Kantwisseling van de aanduidingen (aanduiding totale/partiele deelname valt ook weg):



3. Ontwerpkeuzes

Attribuut verwijst naar ander entiteitstype → relatie

Attribuut bij verschillende entiteitstypes → entiteitstype

Entiteitstype met één attribuut met één entiteitstype relatie → attribuut

Dit laatste moet niet steeds gevolgd worden ivm uitbreidbaarheid.

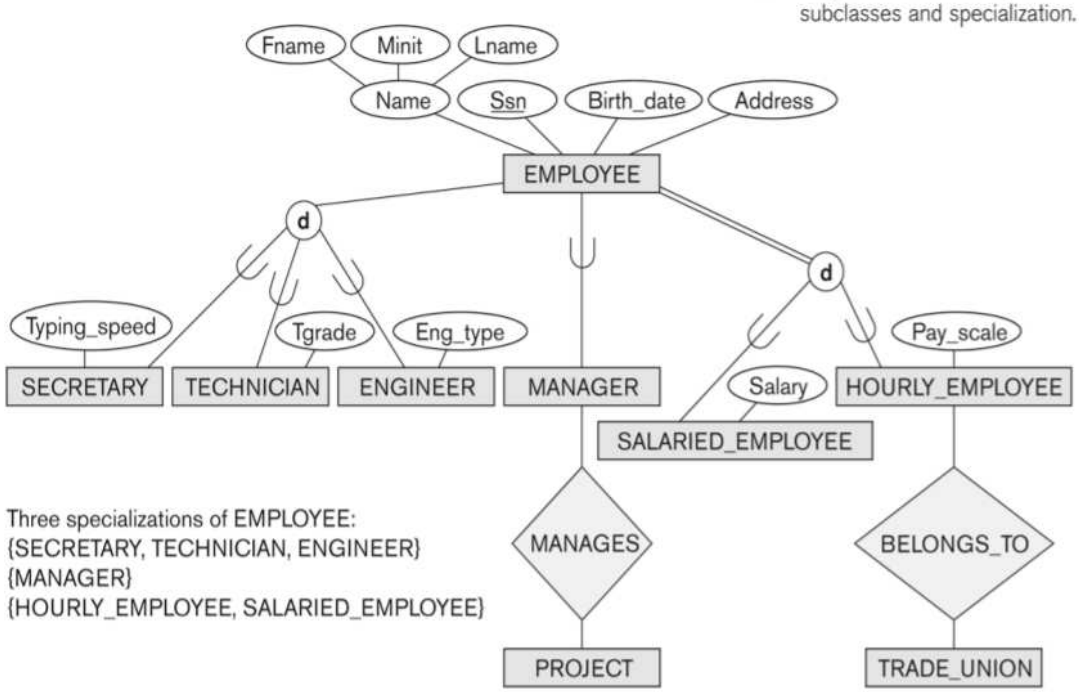
4. Niet- binaire relaties

ALTIJD (min, max) notatie

Een tertiaire relatie kan niet vervangen worden door 3 binaire: informatieverlies

Uitgebreid entiteit relatie model

Figure 4.1
EER diagram notation to represent subclasses and specialization.



1. **Subklasse/superklasse**

Subklasse erft attributen van alle directe en indirecte superklassen! U staat voor deelverzameling (subklasse), d voor disjunct, o voor overlappend (employee zit in meerdere subklassen)

2. **Specialisatie/generalisatie**

Specialisatie: obv predikaat/conditie → predikaatgedefinieerd

obv attribuut → attribuutgedefinieerd

obv andere kenmerken → gebruikergedefinieerd

subklassen (manager, technieker, … van employee) hebben specifieke attributen (typingspeed voor een secretaresse) en zijn disjunct of overlappend (d en o)

enkel bepaalde soorten van werknemers (subklassen) in een relatie (manages tss manager en project)

Totale/partiële specialisatie: totaal → elk object v superklasse moet tot een subklasse behoren

Generalisatie: Omgekeerde: het gemeenschappelijke uit verschillende entiteitstypes

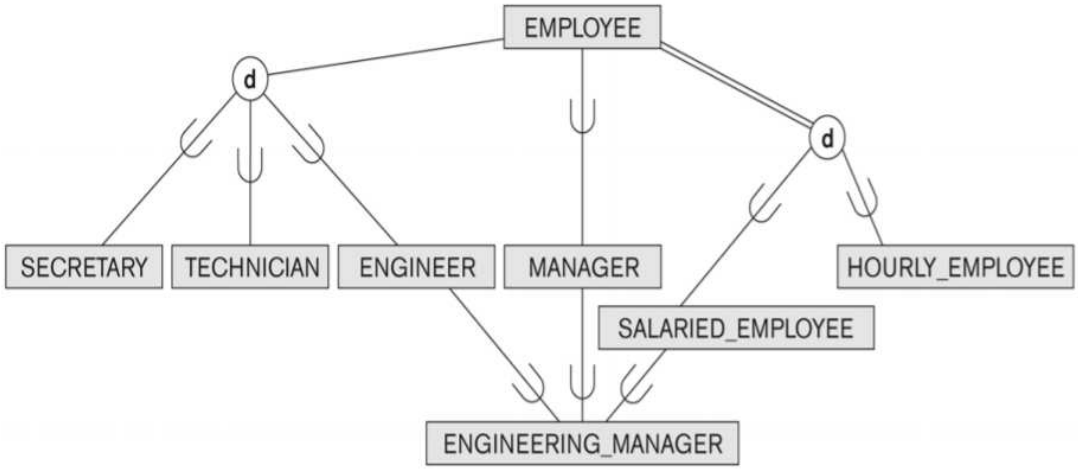
Algemener entiteitstype = superklasse

Levert gewoonlijk totale subklasse/superklasse relatie

Specialisatie-hiërarchie: elke subklasse in één super/subklasse relatie

Specialisatie-tralie: subklasse in meerdere super/subklasse relaties: gemeenschappelijk subklasse

Voorbeeld van gemeenschappelijke subklasse:



Ontwerpmethodes:

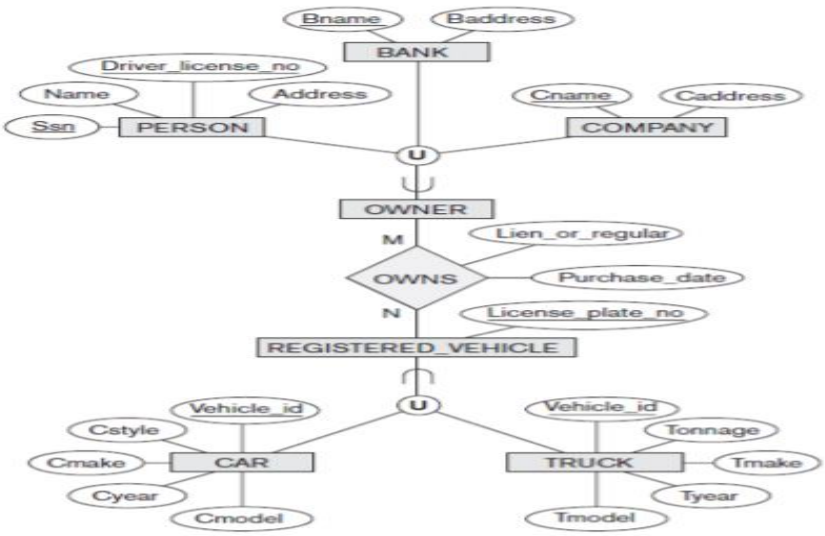
- Top-down ontwerp: begin met 1 entiteitstype, herhaaldelijk specialiseren
- Bottom-up: begin met verscheidene entiteitstypes, herhaaldelijk generaliseren
- Praktijk: combinatie v beiden

3. Categorieën

Categorie (unie type) = subklasse met meerdere superklassen

Categorie vs gemeenschappelijke subklasse:

- Deelverzameling van unie vs deelverzameling van doorsnede
- Entiteit in subklasse behoort tot 1 superklasse vs tot elke superklasse
- Selectieve overerving v attributen vs overerving alle attributen



SAMENVATTING LES 3

Relationeel model en omzetting ER naar relationeel

Relationeel model

Het relationeel model is een implementatiemodel.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

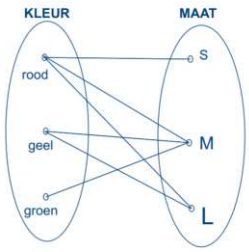
Figure 3.5
Schema diagram for the COMPANY relational database schema.

Een relatie is:
Opgepast:

ER model	Relationeel model
Relationship	*Relation*
Conceptueel niveau	Implementatieniveau
Tussen entiteiten	Over attributen
Ruit in ER diagram	'Tabel' in gegevensbank

Een **relatie** R op de verzamelingen $V_1, ..., V_n$ is een deelverzameling van de productverzameling $V_1 \times ... \times V_n$: $R \subseteq V_1 \times ... \times V_n$

KLEUR	MAAT
rood	S
rood	M
rood	L
geel	M
geel	L
groen	M



Attribuut heeft een naam en een type, is enkelvoudig (atomaire waarden) en de verzameling van alle mogelijke waarden van Ai = DOM(Ai).

tupel over de attributen x = {A1,…,An} →t = { (A1,W1), …, (An, Wn)} met elke Wi ∈DOM(Ai) of NULL (nvt of onbekend)

De tupels en attributen zijn niet geordend, elke tupel komt max 1x voor in een relatie. → uniek
Relaties zijn wel te sorteren obv attributen. Een relatie is een set van tupels.

Notaties:

- Q, R, S: relatienamen
- q, r, s: instanties
- t, u, v, w: tupels

- R.A = attribuut A van relatie R

Regulier entiteitstype → Relatie met alle enkelvoudige attributen, kies sleutelattribuut

Zwak entiteitstype → Relatie met alle enkelvoudige attributen + verwijssleutel (= primaire sleutel van eigenaarsentiteitstype)

Binair 1:1 relatietype → Primaire sleutel van ene als verwijssleutel in andere relatie

Binair 1:N relatietype → aan N-kant primaire sleutel van 1-kant

Binair N:M relatietype → relatie: beide sleutels als verwijssleutels en samen primaire sleutel

Meerwaardig attribuut → relatie met dat attribuut: de primaire sleutel van de bijhorende relatie als verwijssleutel en de primaire sleutel is dan de combinatie van beide

Beperkingen

Intra-relationele restricties: intern binnen 1 relatie, makkelijk te controleren

Inter-relationele restricties: verschillende relaties

1) Domeinrestricties: Beperkt mogelijke waarden van attributen (bv 18-65 voor leeftijd)
2) Sleutelrestricties: Waarden van sleutels moeten uniek zijn
3) Algemene integriteitsrestricties: integriteit = juistheid & volledigheid van GB → statische en dynamische regels

Supersleutel K is een verzameling van attributen die een tupel van R ondubbelzinnig bepalen.

Kandidaatsleutel K is een supersleutel zonder overtollige attributen.

De primaire sleutel: gekozen uit de kandidaatsleutels, de andere zijn alternatieve sleutels

Statische regels (slaan op vaste toestand GB, gelden in alle mogelijke extensies):

- Attribuutrestricties: domein is enkelvoudig (eis van de 1^e NV)
- Entiteit-restricties: Tupel uniek in extensie, geen null-waarden in primaire sleutel
- Referentiële integriteit: als tupel verwijst naar ander tupel, moet dat tupel bestaan
- Verzameling attributen FK v R1 is verwijssleutel asa:
    - Attributen v FK hebben zelfde domein als primaire sleutelattributen PK v R2
    - Elke waarde v FK in R1 komt voor als waarde v/e tupel in R2 of is null
- Verwijssleutel kan naar eigen relatie verwijzen (recursieve relatie)
  Bv SUPERSSN (overste v werknemer is ook werknemer)

Dynamische regels (slaan op toestandsovergangen, gelden ∀ overgangen v 1 extensie naar andere)

- Autorisatieregels: Mag de gebruiker die actie uitvoeren?
- Coördinatieregels: Bv extra tupel wanneer saldo < 0
- Precondities/postcondities: geld afhalen kan enkel als saldo > 0
- Overgangsregels: Beschrijven welke volgordes van extensies zijn toegelaten

Gegevens aanpassen (toevoegen, weglaten, wijzigen) → Integriteit controleren

Bv. Een cursus verwijderen → Ook 'volgt' aanpassen bij studenten

(E)ER → relationeel schema

ER → Relational

1) Non-weak entities
8) Sub/Super classes
2) Weak entities
3) 1:1 relations
4) 1:N (non-identifying) relations
5) N:M relations
6) Multi-valued attributes
7) Ternary+ (non-identifying) relations

ER model	Relationeel model
entiteitstype	"entiteits" relatie
1:1 of 1:N relationship type	verwijssleutel of relatie
M:N relationship type	relatie met 2 verwijssleutels
n-aire relationship type	relatie met n verwijssleutels
enkelvoudig attribuut	attribuut
samengesteld attribuut	verzameling attributen
meerwaardig attribuut	relatie of verwijssleutel
value set	domein
sleutelattribuut	primaire sleutel

Stap 1: ∀ reguliere entiteitstype E: maak R met alle enkelvoudige attributen v E, kies PK

! Opgelet: Stap 8 uitvoeren voor stap 2.

Stap 2: ∀ zwak entiteitstype W: maak R met alle enkelvoudige attributen v W, PK v eigenaar als verwijssleutel

Stap 3: ∀ binair 1:1 R: PK van T als verwijssleutel van S of omgekeerd, vermijd null-values (totale participatie), neem enkelvoudige attributen op

! Opgelet: stap 4 niet toepassen op de definiërende relatie v/e zwakke entiteit (reeds stap 2)

Stap 4: ∀ binair 1:N R: PK v 1-kant aan de N-kant toevoegen, neem enkelvoudige attributen op

Stap 5: ∀ N:M R: nieuwe relatie S met als verwijssleutels de PKs de deelnemende entiteitstypes, de verwijssleutels samen = PK v S, neem enkelvoudige attributen van R op

Stap 6: ∀ meerwaardig attribuut A: maak R met attribuut = A + PK K van bijhorende relatie als verwijssleutel, A en K samen is dan de PK van R

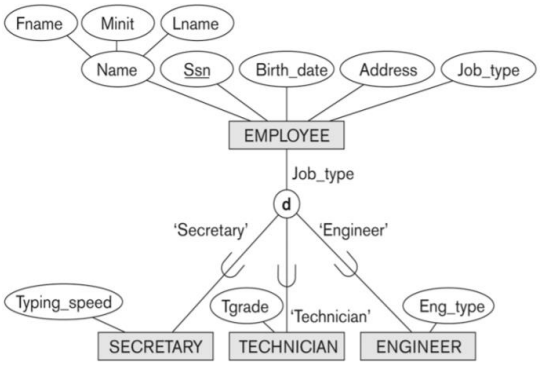
Stap 7: ∀ n-air relatietype R: maak nieuwe relatie S met als verwijssleutels de PKs van de deelnemende entiteitstypes en met alle enkelvoudige attributen van R, de PK = alle verwijssleutels samen (zie gelijkenis met stap 5)

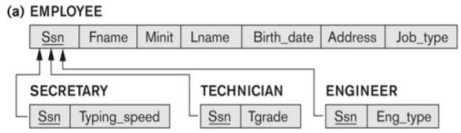
Stap 8: behandeling van super/subklasse-relaties

Elke specialisatie met m subklassen { S1, S2,..., Sm } en superklasse C met attributen { k, a1, ..., an }

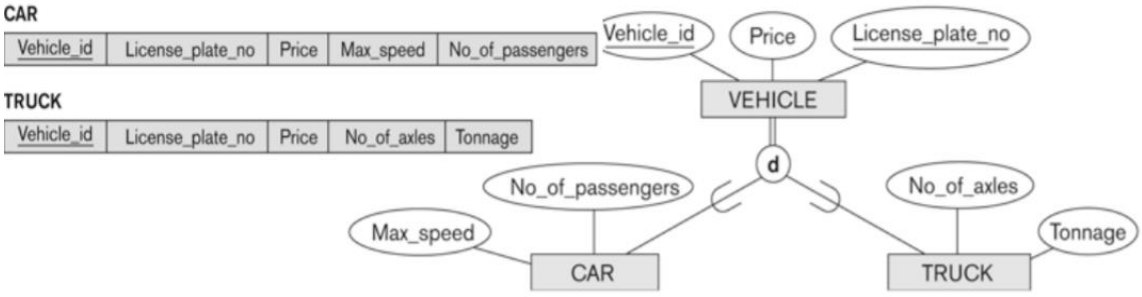
- 8A: - maak relatie L voor C met attr(L) = {k, a1, …, an} en PK(L) = k

  - en relaties Li ∀ subklasse Si met attr(Li) = {k} ∪ {attr(Si)} en Pk(Li) = k

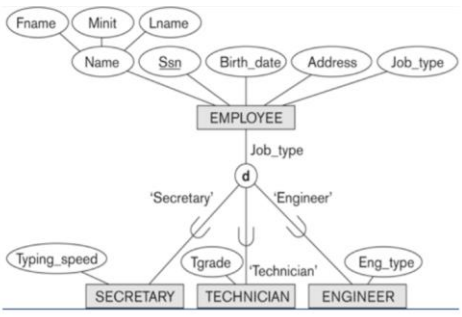




- 8B: - maak ∀ subklasse Si relatie Li met attr(Li) = {attr(Si)} ∪ { k, a1, ..., an } en PK(Li) = k
  - werkt enkel correct voor disjuncte, totale specialisatie
  - niet totale specialisatie → verlies van gegevens (∉ Si)
  - niet disjuncte specialisatie → redundantie

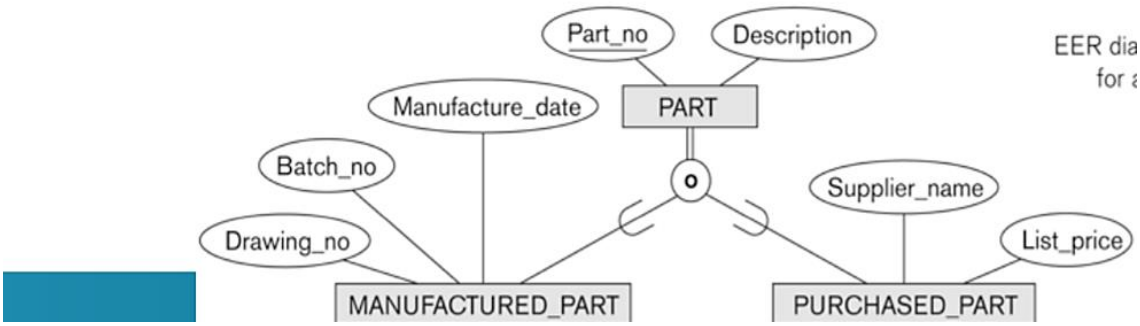


- 8C: één relatie L met attr(L) = {k,a1, ..., an} ∪ {attr(S1)} ∪...∪ {attr(Sm)} ∪ {t} PK(L) = k
  waarbij t aanduidt tot welke subklasse elk tupel behoort:
  – t weggelaten als specialisatie predikaatgedefinieerd is
  – enkel voor disjuncte subklassen
  – veel verschillende attributen in subklassen: veel null
  - ➔ dus vnl. gebruikt als er niet veel subklasse-attributen zijn
  - ➔ er is geen equijoin of uitwendige join nodig



- 8D: één relatie L met attr(L) = { k, a1, ..., an } ∪ { attr(S1) } ∪ ... ∪ { attr(Sm) } ∪ { t1, t2, ..., tm } en PK(L) = k
  – ti , 1≤ i ≤ m is boolean die aangeeft of entiteit in Si zit
  gebruikt voor overlappende subklassen, ook hier gevaar voor veel nulwaarden

Part_no	Description	Mflag	Drawing_no	Manufacture_date	Batch_no	Pflag	Supplier_name	List_price



SAMENVATTING LES 4

woensdag 13 mei 2020 9:47

Relationele algebra

Cheat sheet

Relational Algebra

$\sigma_{criteria}(R)$	: selection
$\pi_{attributes}(R)$	: projection
$R \cup S$	: union
$R - S$	: minus
$R \times S$	: cartesian product
$R \bowtie_{criteria} S$	: join = $\sigma_{criteria}(R \times S)$
$R * S$	: natural join on common columns
$R(X,\ldots) *_{X,Y} S(Y,\ldots)$	: natural join = $R \bowtie_{X == Y} S$
$R \overrightarrow{\bowtie} S$	: left outer join (with nulls in S)
$R \overleftarrow{\bowtie} S$	: right outer join
$R \overleftrightarrow{\bowtie} S$	: full outer join
$\rho_{S(attributes)}(R)$	: renaming
$R \cap S$	: intersection
$R \cup^{+} S$	: full union
$R \cup^{-} S$	: union with common attributes only
$R \div S$	: division
${}_{attr}\mathcal{F}_{aggr\ attr}(R)$	: aggregation (count,sum, avg, min, max)

Eigenschappen relationeel model

• **relatie**: een tabel met kolommen en rijen
• **attribuut**: een kolom van een relatie
• **tupel**: een rij van een relatie.
• **domein**: verzameling van toegelaten waarden voor 1 of meerdere attributen. Voorbeelden zijn “integers” , “strings” , “datum”, maar ook “postcodes”, …

• Attribuut-waarden zijn altijd atomair
• Een relatie is een verzameling tupels
• Iedere relatie heeft een primaire sleutel

Algebraïsche talen

– steunen op relationele algebra
– gebruiken operatoren
– proceduraal; **HOE**

• verzamelingsoperatoren
  – unie, doorsnede, verschil, cartesisch product
• relationele operatoren
  – selectie, projectie, join, deling, hernoeming
• unaire en binaire operatoren

Selectie

$$\sigma_{<selectiecriterium>}(R)$$

=> selecteert een aantal tupels uit een extensie (= rijen uit een tabel)

• resultaat: – een relatie (tabel) met hetzelfde schema
  – deelverzameling van de oorspronkelijke extensie
• selectiecriterium F = (logische) formule
  – enkelvoudige formules: =, ≠, , <=, >=
  – meervoudige formules: logische operatoren ∧, ∨, ¬

Voorbeelden:

$\sigma_{DNO=4}$ (EMPLOYEE)

$\sigma_{SALARY > 30000}$ (EMPLOYEE)

$\sigma_{(DNO = 4 \wedge SALARY > 25000) \vee (DNO = 5 \wedge SALARY > 30000)}$(EMPLOYEE)

Eigenschappen:
• behoudt het schema
• σ(r) ⊆ r; dus kardinaliteit stijgt niet: #(σ(r)) ≤ #r
• samenstelling van selecties is commutatief

Projectie

$\pi_{<attributenlijst>}(R)$

=> doel: een aantal kolommen uit een tabel halen
- resultaat: - verzameling tupels
  - met attributen deelverzameling van attributen van oorspronkelijke tupels
  - verbonden met deelverzameling X van het tupelschema

Voorbeelden:

$\pi_{FNAME, LNAME, SALARY}$ ($\sigma_{DNO = 5}$ (EMPLOYEE))

Eigenschappen:
- #π_X (R) ≤ # r
  - Reden: dubbelen worden verwijderd
  - #π_X (R) = # r indien X een sleutel bevat
- π_X (π_Y (r)) enkel gedefinieerd indien X ⊆ Y
  - Dus niet commutatief!!!
- Idempotent : π_{X1} (π_{X1} (R)) = π_{X1} (R)
- Enkel allerlaatste(buitenste) projectie moet uitgevoerd worden

Hernoeming

RESULT(Firstname, Lastname, Salary)

$\leftarrow \pi_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)

=> doel: wijzigen van de attribuutnamen
- Notatie: nieuwe namen tussen haakjes vermeld
- Kan ook met de operator ρ voorgesteld worden

1. $\rho_{S(B1, B2,..., Bn)}$ (R) relatie en attributen worden hernoemd
2. ρ_{S} (R) alleen relatie wordt hernoemd
3. $\rho_{(B1, B2,..., Bn)}$ (R) alleen attributen worden hernoemd

Voorbeelden:

$\rho_{TEMP}(\sigma_{DNO=5}$ (EMPLOYEE))

$\rho_{R(First\_name,Last\_name,Salary)}\pi_{FNAME,LNAME,SALARY}(TEMP)$

Unie, doorsnede en verschil

- Enkel toegelaten op vergelijkbare ("union compatible") relaties

2 relaties R(A1, …, An) en S(B1,…,Bm) zijn vergelijkbaar als en slechts als
- n = m (d.w.z. R en S hebben dezelfde graad)
- DOM(Ai) = DOM(Bi) voor 1 ≤ i ≤ n

- Schemabehoudend, op attribuutnamen na
  - Afspraak: behoud attribuutnamen van 1ste relatie

Voorbeeld : RESULT <-- RESULT1 U RESULT2

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Cartesisch product

Relaties: R(A1, …, An) en S(B1,…,Bm) die niet noodzakelijk vergelijkbaar zijn
Q = R x S
- Heeft een schema Q(A1, …, An, B1, …, Bm)
- Bevat elke combinatie van tupels uit R en S

Voorbeeld:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

R1 X S1 =

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/
22	dustin	7	45.0	58	103	11/12/
31	lubber	8	55.5	22	101	10/10/
31	lubber	8	55.5	58	103	11/12/
58	rusty	10	35.0	22	101	10/10/
58	rusty	10	35.0	58	103	11/12/

Join operator

$$R \bowtie_F S$$

- F = selectiecriterium
- Binaire operator
  - Combineert gerelateerde tupels van 2 relaties
- = cartesisch product + selectie

$$R \bowtie_F S = \sigma_F (R \times S)$$

Voorbeeld:

$$\text{DEPARTMENT} \bowtie_{\text{MGRSSN = SSN}} \text{EMPLOYEE}$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	···	Fname	Minit	Lname	Ssn	···
Research	5	333445555	···	Franklin	T	Wong	333445555	···
Administration	4	987654321	···	Jennifer	S	Wallace	987654321	···
Headquarters	1	888665555	···	James	E	Borg	888665555	···

Soorten joins

- Opdeling naargelang van de vorm van de join-voorwaarde F
  - F = C1 ∧ C2 ∧ … ∧ Cn
- **Theta-join**: elke Ck is van de vorm Ai θ Bj met
  - θ ∈ { = , <, >, ≤, ≥, ≠}
  - DOM(Ai) = DOM(Bj)
- **Equi-join**: enkel "="
- **Natuurlijke join**
  - Equi-join + weglaten van overtollige attributen die dezelfde naam hebben
    (d.w.z. per voorwaarde Ai = Bj enkel Ai of Bj behouden)
  - Notatie: $R *_F S$

Vereenvoudigde notaties voor natuurlijke join

- $R *_{X,Y} S$
  - Met X en Y attribuutlijsten (van gelijke lengte)
  - X1 = Y1 ∧ … ∧ Xk = Yk
  - Alleen attributen in X blijven behouden na join

- R * S
  - Lijsten X en Y zijn impliciet: bevatten alle attributen die dezelfde naam hebben in R en S

Voorbeeld:

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

R1 ∗ S1 =

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

Fundamentele operatoren

- De verzameling operatoren {σ, π, U, - , x} is volledig:
  - Andere operatoren kunnen op basis van deze gedefinieerd worden

- Niet- fundamentele operatoren:
  - Join
  - ∩
  - Deling
  - …

    => niet strikt nodig, wel makkelijk

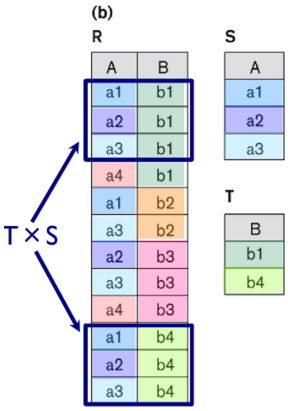
Deling

T = R ÷ S

=> = inverse van cartetisch product
- ⇔ T is de maximale relatie waarvoor geldt dat T x S ⊆ R
- T bevat enkel de attributen van R die niet in S zitten<

=> berekent in 1 operatie of een gegeven verzameling waarden samen voorkomt voor combinaties van waarden voor de overige attributen

Voorbeeld:



Aggregaatfuncties

groepering $\Im$ functies (R)

- Groepering = verz. Attributen op basis waarvan groepering gebeurt
- Functies = lijst van koppels (functie, attribuut)

=> Functies die op verzamelingen waarden uitgevoerd worden
- SUM, AVERAGE, MAX, MIN, COUNT

=> Resultaat: tabel met als attributen
- Attributen uit de groepering
- Attributen met naam "functie_attribuut" die het resultaat van de functie op dat attribuut geven

=> groeperingsattributen kunnen weggelaten worden
- aggregaatfunctie toegepast op hele relatie -> resultaat: 1 tupel

Voorbeeld:

$$Dno\ \mathfrak{I}\ COUNT\ Ssn, AVERAGE\ Salary\ (\ EMPLOYEE\ )$$

! Let op de juiste groepering

Recusieve sluiting

- Vb1
  - Vind alle ondergeschikten van persoon Y
  - = transitieve sluiting van "X heeft Y als chef"
- Vb2
  - In een stamboom-databank van honden:
  - Vind alle voorouders van "zwerver"

- Niet algemeen uit te drukken in relationele algebra:
  - Wel voor bepaald aantal niveaus, bv. Vind alle ouders, vind alle grootouders…
  - Relationele uitdrukking groeit per niveau dat erbij komt
  - Onbeperkt aantal niveaus zou oneindige uitdrukking geven

Uitwendige join

- Gewone joins
  - Leveren over de tupels die niet aan de join voorwaarde voldoen geen enkele informatie op
  - Vb.
    - Lijst van alle werknemers + als ze een departemant leiden: naam van dat departement
    - Join geeft enkel werknemers die effectief een departement leiden

- Linkse uitwendige join (LEFT OUTER JOIN)
  - Levert info over alle werknemers
  - + info over het departement dat ze leiden, of nul als ze geen departement leiden

- Analoog
  - Rechtse uitwendige join
  - Volledige uitwendige join

Voorbeeld: LEFT OUTER JOIN

$$TEMP \leftarrow (\ EMPLOYEE\ \bowtie_{Ssn = Mgr\_ssn}\ DEPARTMENT\ )$$

$$RESULT \leftarrow \pi_{\ Fname, Minit, Lname, Dname}\ (\ TEMP\ )$$

Varianten op unie

- Vereniging van tupels van niet-vergelijkbare relaties

- Uitwendige unie: U^+
  - Notatie: Q = R U^+ S
  - Attr(Q) = Attr(R) U Attr(S)
  - Nulwaarde voor alle attributen die niet van toepassing zij

- Inwendige unie: U^-
  - Notatie: Q = R U^- S
  - Attr(Q) = Attr(R) ∩ Attr(S)

Samenvatting les 5

donderdag 14 mei 2020 11:46

SQL 1

Wat is SQL?

Calculustalen:
- Beschrijving gewenste info (adhv relationele calculus)
- Declaratief: **WAT**

declarativiteit →

SQL:
- Vooral declaraties (WAT)

→ Voor verwerking en optimalisatie

Algebraïsche talen:
- Operatoren & relationele algebra
- Proceduraal: **HOE**

Terminologie:

Relationeel model	SQL
Schema	Schema
Relatie →	Tabel
Tupel	Rij
Attribuut	Kolom

Datatypes

Numeriek

Gehele waarden:
- INTEGER - INT
- SMALLINT

Niet-gehele waarden:
- FLOAT - REAL
- DOUBLE PRECISION

Geformatteerd:
- DEC (i,j) -- NUMERIC (i,j)
- i= precisie (# decimale cijfers) ;
- j = schaal (# cijfers na dec. Punt)

Karakters

Vaste lengte:
- CHAR(n)
- CHARACTER(n)

Variabele lengte:
- VARCHAR(n)
- CHAR(ACTER) VARYING(n)

Syntax:
- "abc" of 'abc'

Datum en tijd

DATE:
- Vb. DATE'2004-02-23'

TIME:
- Vb. TIME'09:12:47'

TIMESTAMP:
- Vb. TIMESTAMP'2004-02-23 09:12:47 648302'

Opm: mogelijk nog verschillende namen, types, syntax

Schema's creëren en aanpassen

Catalogus

(= verzameling schema's in een SQL omgeving)

bevat:
- SQL schema's incl één speciaal: INFORMATION_SCHEMA
  - Schema's kunnen zo elementen delen
- Mogelijke restricties

Schema

bevat:
- Schemanaam
- eigenaar ({AUTHORIZATION *<name>*})
- beschrijving van elementen (tabellen, constraints, views, domeinen, autorisaties,…)

```
CREATE SCHEMA COMPANY
    AUTHORIZATION Jsmith ;
```

Tabel

Creatie in schema in SQL

```
CREATE TABLE  [ <schema name >. ] <table name>
    ( { <column name> <column type> [<attribute constraint>] }
     {<table constraint>} *)
```

Vb.

```
CREATE TABLE EMPLOYEE
      ( Fname                 VARCHAR(15)          NOT NULL,
        Minit                 CHAR,
        Lname                 VARCHAR(15)          NOT NULL,
        Ssn                   CHAR(9)              NOT NULL,
        Bdate                 DATE,
        Address               VARCHAR(30),
        Sex                   CHAR,
        Salary                DECIMAL(10,2),
        Super_ssn             CHAR(9),
        Dno                   INT                  NOT NULL,
      PRIMARY KEY (Ssn),
      FOREIGN KEY(Super_ssn) REFERENCES EMPLOYEE(Ssn),
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Wijzigen van tabel

```
ALTER TABLE <name>
    {  ADD [COLUMN] <column name> <column type>
     | DROP [COLUMN] <column name> <drop behaviour>
     | ALTER [COLUMN] <column name> <default option>
     | ADD  CONSTRAINT    <constraint definition>
     | DROP  CONSTRAINT  <constraint name>  <drop
                              behaviour> }
```

- <drop behaviour> =
  - CASCADE | RESTRICT
- <default option> =
  - DROP DEFAULT | SET DEFAULT <value>

Vb.

```
ALTER TABLE  COMPANY.DEPARTMENT
      ALTER COLUMN Mgr_ssn SET DEFAULT '33445555' ;
```

Attribuut

Definiëren:

ofwel rechtstreeks

ofwel via DOMAIN:

```
CREATE DOMAIN  SSN_TYPE AS CHAR(9) ;
```

Restricties

Definiëren van restricties op een tabel

Soorten:

Primaire sleutel: PRIMARY KEY *<attrlist>*

Alternatieve sleutel: UNIQUE *<attrlist>*

Verwijssleutel: FOREIGN KEY *<attrlist>* REFERENCES *<tabel><attrlist>*

Actie bij schending van referentiële integriteit:

Trigger --> ON DELETE/ ON UPDATE

Actie--> SET NULL / SET DEFAULT / CASCADE / RESTRICT

```
CREATE TABLE EMPLOYEE
      ( ...,
        Dno             INT         NOT NULL        DEFAULT 1,
      CONSTRAINT EMPPK
        PRIMARY KEY(Ssn),
      CONSTRAINT EMPSUPERFK
        FOREIGN KEY(Super_ssn) REFERENCES EMPLOYEE(Ssn)
                    ON DELETE SET NULL        ON UPDATE CASCADE,
      CONSTRAINT EMPDEPTFK
        FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                    ON DELETE SET DEFAULT     ON UPDATE CASCADE );
```

Definiëren van restricties op een attribuut

NOT NULL (automatisch voor primary key's)

DEFAULT *<value>*
CHECK (voorwaarde)

```
Dnumber INT NOT NULL CHECK (Dnumber > 0 AND
                            Dnumber < 21);
```

Specificeren van restricties

2 opties:
- Direct bij tabel / attribuut (zie hoger)
- Met sleutelwoord 'CONSTRAINT' + naam:

```
CREATE TABLE EMPLOYEE
    ( ...,
      Dno          INT        NOT NULL      DEFAULT 1,
    CONSTRAINT EMPPK
      PRIMARY KEY(Ssn),
```

Verwijderen van tabel/schema

Drop behaviour:

Schema	Tabel
DROP SCHEMA *<name> < drop behaviour>*	DROP TABLE *<name> < drop behaviour>*
CASCADE	CASCADE
Schema + alle tabellen verwijderen	Gerefereerde restricties en views ook verwijderen
RESTRICT	RESTRICT
Schema verwijderen als het leeg is	Tabel verwijderen als er geen referenties/views naar verwijzen

Gegevens selecteren en projecteren

Gegevens opvragen mbv queries

SQL queries	**Relationele algebra**
SELECT *<attributen lijst>*	'projectie'
FROM *<tabellen lijst>*	'cartesisch product'
WHERE *<condities>*	'selectie'

Vb. ⟷

SELECT bdate
FROM employee
WHERE **employee.**Fname = 'John'

$\pi_{bdate}(\sigma_{Fname = 'John'}(employee)$

(Gebruiken indien dubbelzinnige attribuutnamen)

SELECT *
- Geen projectie

WHERE weglaten
- Geen selectie

Joins opbouwen

Impliciete	**vs**	**expliciete join notatie**
SELECT Pname, Essn **FROM** PROJECT, WORKS_ON **WHERE** Dnum=5 **AND** Pno=Pnumber;		**SELECT** Pname, Essn **FROM** (PROJECT **JOIN** WORKS_ON **ON** Pno=Pnumber) **WHERE** Dnum=5;

Andere voorbeelden:
- NATURAL JOIN
- LEFT/RIGHT OUTER JOIN

Aliasen

Na sleutelwoord AS of na de naam van de relatie

```
SELECT   E.Fname,  E.Lname, S.Fname, S.Lname
FROM     EMPLOYEE  AS E,   EMPLOYEE AS S
WHERE    E.Super_ssn = S.Ssn ;
```

```
SELECT   E.Fname,  E.Lname, S.Fname, S.Lname
FROM     EMPLOYEE  E  S
WHERE    E.Super_ssn = S.Ssn ;
```

Verschil tussen tabellen en relaties

Relatie uit relationeel model = verzameling/set van tupels ==> geen dubbels
Tabellen uit SQL = multiset van tupels ==> wel dubbels mogelijk+
--> SELECT DISTINCT
Terzijde: impliciet doet SQL SELECT ALL als er SELECT staat

Operatoren

Verzamelingsoperaties

UNION / INTERSECT / EXCEPT(=verschil)
Vb.

– Q 4: geef alle projecten waarbij Smith betrokken is als manager van het controlerend departement of waaraan hij meewerkt

```
(SELECT    DISTINCT Pnumber
 FROM      PROJECT, DEPARTMENT, EMPLOYEE
 WHERE     Dnum = Dnumber AND Mgr_ssn = Ssn AND Lname = 'Smith')

UNION

(SELECT    DISTINCT Pnumber
 FROM      PROJECT, WORKS_ON, EMPLOYEE
 WHERE     Pno = Pnumber AND Essn = Ssn AND Lname = 'Smith') ;
```

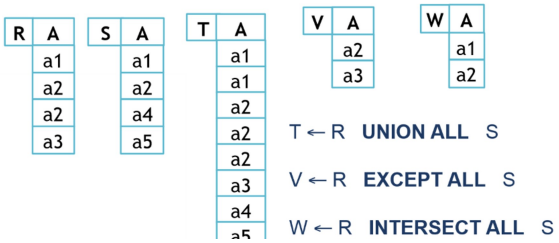
Opmerkingen:

- Relaties moeten vergelijkbaar (= union compatible) zijn
  - Zelfde attribuuttypes (zoals bij relationele algebra) in zelfde volgorde (extra tov relationele algebra)
- Deze operaties elimineren wel automatisch dubbelen (zoals bij relationele algebra)

Operaties met multisets

UNION ALL / EXCEPT ALL / INTERSECT ALL
(hier wel expliciet ALL bij schrijven)

- sleutelwoord ALL na de operator:
  - UNION ALL
  - EXCEPT ALL
  - INTERSECT ALL



String- en rekenkundig

String

Patroon { LIKE }
% = aantal karakters ; _ = één karakter

```
SELECT   Fname,Lname
FROM     EMPLOYEE
WHERE    Address LIKE  '%Houston,TX%' ;
```

```
SELECT   Fname,Lname
FROM     EMPLOYEE
WHERE    Bdate LIKE  '_ _5_ _ _ _ _ _ _ ' ;
```

Reken

+ - * /
Op numerieke waarden
Verschil tussen data/tijden

Concatenatie { || }

Anderen: between, order by, null

BETWEEN

```
SELECT    *
FROM      EMPLOYEE
WHERE     (Salary BETWEEN 30 000 AND 40 000) AND  Dno = 5 ;
```

ORDER BY *<attributen>* [ASC | DESC] (--> SQL-opdracht genoeg gedaan)

NULL

Alle employees zonder supervisor

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     Super_ssn IS NULL ;
```

Aggregaatfuncties

= COUNT , SUM , MAX , MIN , AVG

COUNT te gebruiken met:

- *
- DISTINCT

SELECT

Ieder geselecteerd attribuut = in combinatie met de gebruikte aggregaatfunctie of met de GROUP BY

GROUP BY

(zonder GROUP BY slechts 1 tupel als resultaat)

Vb

```
SELECT      Dno, COUNT(*), AVG(Salary)
FROM        EMPLOYEE
GROUP BY    Dno ;
```

link relationele algebra

Dno ℑ COUNT Ssn, AVERAGE Salary (EMPLOYEE)

HAVING

Alleen in combinatie met GROUP BY

Evaluaart waarden van aggregaatfunctie per groep

Tupels toevoegen, wijzigen en verwijderen

INSERT UPDATE DELETE

INSERT

Impliciete vs expliciete vorm

Zonder expliciete vermeldig van attributen (volgorde schema behouden!)

```
INSERT INTO     EMPLOYEE
VALUES          ('Richard', 'K', 'Marini', '653298653',
                '1962-12-30', '98 Oak Forest, Katy', TX',
                'M', 37000, '987654321', 4) ;
```

Expliciete vermelding (eigen volgorde mogelijk)

```
INSERT INTO     EMPLOYEE (Fname, Lname, Ssn)
VALUES          ('Richard', 'Marini', '653298653') ;
```

Null bij niet vermelde waarden

Vaak direct na creatie tabel (= laden van resultaat van een query)

```
CREATE TABLE        DEPTS_INFO
      ( Dept_name           VARCHAR(15),
        No_of_emps          INTEGER,
        Total_sal           INTEGER );

INSERT INTO         DEPTS_INFO (Dept_name,No_of_emps,Total_sal)
      SELECT            Dname, COUNT(*), SUM(Salary)
      FROM              ( DEPARTMENT JOIN EMPLOYEE ON
                        Dnumber = Dno )
      GROUP BY Dname ;
```

Tabel van dit voorbeeld is niet zo'n goed idee

redundantie + mogelijk verlies van integriteit

beter : view definiëren (zie volgende les)

UPDATE

UPDATE *<tabel>*

SET { *<attr>* = *<expr>* , … }

WHERE *<conditie>*

DELETE

DELETE FROM *<tabel>*

WHERE *<conditie>*

Geen where? Alle tupels weg

Samenvatting les 6
donderdag 14 mei 2020 11:46

SQL 2

Views

Views

View = afgeleide relatie: tupels niet expliciet opgeslagen maar berekend uit andere relatie (=de definiërende tabellen v/d view)
m.a.w. view == VIRTUEEL
== NIET GEMATERIALISEERD
Kan analoog aan 'echte' tabellen SELECT op toegepast worden

Syntax:
CREATE VIEW *<viewname>[<lijst van gewenste attribuutnamen>]* (== optioneel, indien weggelaten gewoon geselecteerde namen uit query)
AS *<query>*
Vb.

```
CREATE VIEW   WORKS_ON1
AS  SELECT    Fname, Lname, Pname, Hours
    FROM      EMPLOYEE, PROJECT, WORKS_ON
    WHERE     Ssn = Essn  AND  Pno = Pnumber ;
```

Voordelen:
Eenvoudigere queries
Beveiligingsmechanisme --> gebruiker krijgt zicht op deel van de gegevensbank
View heeft t.o.v. nieuwe tabel aanmaken geen reduntantie en blijft up-to-date

Verwijderen

DROP VIEW *<viewname>*;
Opmerking: er worden hierbij geen tupels verwijderd (want virtueel & niet-gematerialiseerd)

2 benaderingen voor implementatie

Twee manieren om om te gaan met query op een view (manier 2 > manier 1)

1 Query modification
Query op view ⟶ query op onderliggende tabellen

– de query QV1 op view WORKS_ON1

```
SELECT  Fname, Lname
FROM    WORKS_ON1
WHERE   Pname = 'ProductX' ;
```

– wordt automatisch omgevormd tot

```
SELECT  Fname, Lname
FROM    EMPLOYEE, PROJECT, WORKS_ON
WHERE   Ssn = Essn  AND  Pno = Pnumber
                    AND  Pname = 'ProductX' ;
```

Nadeel:
- Mogelijks complexe en tijdsintensieve queries

2 view materialization
Tijdelijke creatie (=materialisatie) van afgeleide tabel wanneer ze voor het eerst gebruikt wordt
Voordelen:
- Vereist slechts 'incremental update' wanneer basistabellen wijzigen: aanpassing afgeleide tabel ipv geheel nieuwe tabel **(performantie ↗)**
- Wordt automatisch verwijderd indien tijdje niet gebruikt

Wijzigen van view

View = afgeleid uit andere relatie ⇒ wijziging moet doorgegeven worden aan die relaties
Opmerking:
- Indien {VIEW ⟵ 1 basisrelatie} ⟶ aanpassing mogelijk als de view een primaire sleutel / kandidaatsleutel van die basisrelatie bevat
  ⇒ te wijzigen tupel is eenduidig bepaald: aanpassing view = aanpassing basisrelatie
- Indien {VIEW ⟵ join van meerdere basisrelaties} ⟶ meestal niet aanpasbaar **== ambigue wijzigingen**
  Gebruiker keuze laten maken of DBMS meest waarschijnlijke laten kiezen

Vb.

Wijziging op view:

```
UPDATE    WORKS_ON1
SET       Pname = 'ProductY'
WHERE     Lname = 'Smith' AND Fname = 'John'
          AND Pname = 'ProductX' ;
```

Optie 1: naam van project wijzigt

Optie 2: John Smith werkt nu aan ProjectY ipv ProjectX

```
UPDATE    PROJECT
SET       Pname = 'ProductY'
WHERE     Pname = 'ProductX' ;
```

```
UPDATE    WORKS_ON
SET       Pno = (SELECT Pnumber FROM PROJECT
                 WHERE Name = 'ProductY')
WHERE     Essn =    (SELECT Ssn FROM EMPLOYEE
                     WHERE Lname = 'Smith' AND Fname = 'John')
          AND
          Pno IN    (SELECT Pnumber FROM PROJECT
                     WHERE Pname = 'ProductX') ;
```

- Resultaten van aggregaatfuncties kunnen niet aangepast worden
  zouden ook zinloos/betekenisloos zijn: vb. UPDATE DEPTS_INFO
  SET TOTAL_SAL = 100 000 **-- betekenis?**
  WHERE Dname = 'Research;

Geneste queries

Tupel IN verzameling

```
SELECT DISTINCT   Essn
FROM      WORKS_ON
WHERE     (Pno,Hours) IN
               (SELECT     Pno,Hours
               FROM        WORKS_ON
               WHERE       Essn=`123456789') ;
```

ALL, ANY

Vergelijking met verzameling waarden (>, <, ≥, ≤, =, ≠)
ALL ⟶ conditie moet voor alle waarden voldoen
ANY⟶ conditie moet voor ten minste 1 waarde voldoen

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     Salary > ALL
               (SELECT     Salary
               FROM        EMPLOYEE
               WHERE       Dno=5) ;
```

Gecorreleerde geneste queries

Binnenste query afhankelijk van buitenste
⟶ binnenste moet voor elke waarde van de buitenste worden uitgevoerd == DURE operatie

```
SELECT    E.Lname, E.Fname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN
               (SELECT     Essn
               FROM        DEPENDENT AS D
               WHERE       E.Fname = D.Dependent_name
                           AND E.Sex = D.Sex) ;
```

ja, de waarden van dit attribuut zijn voornamen

Opmerking: in binnenste/geneste query kunnen attributen van de buitenste query gebruikt worden. Andersom niet!

EXISTS

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     NOT EXISTS
               (SELECT     *
               FROM        DEPENDENT
               WHERE       Ssn=Essn) ;
```

Permissies

Geven

GRANT *<rechten[(attrlijst)]>* vb.: Insert, delete, select, update(salary)
ON *<schema,tabel>* Employee, department
TO *<gebruiker>;* UserX
(WITH GRANT OPTION)

Ontnemen ⟷ geven

REVOKE *<rechten[(attrlijst)]>*
ON *<schema,tabel>*
FROM *<gebruiker>;*

Permissies op views

Analoge syntax als voor 'echte tabellen'
Ook mogelijkheid om VIEWS READ-ONLY te maken

Restricties (= assertions) en triggers

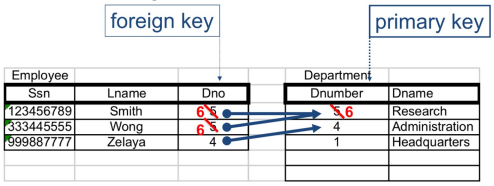
Restricties

Ter herinnering

Restricties mogelijk op…

… tabellen

- Primaire / alternatieve sleutel (PRIMARY / UNIQUE KEY)
  gecheckt bij toevoegen/wijzigen van tupels
- Verwijssleutel (FOREIGN KEY … REFERENCES)
  gecheckt bij toevoegen/wijzigen van tupels in tabel
  gecheckt bij **verwijderen**/wijzigen van tupels in **verwijzende** tabel
- Referentiële integriteit



… attributen

- NOT NULL -- DEFAULT <*value*> -- CHECK (conditie)
  Gecheckt bij toevoegen/wijzigen van attribuutwaarden

In relationele model

Intra- of inter-relationele restricties (één of meerdere relaties betrokken)

Andere restricties dan sleutel-, entiteits- en referentiële restricties:

- **ASSERTIONS**

Gecheckt bij creatie van de assertion + elke latere verwijzing
Opmerking: kan veel extra werk vereisen
Vb

```
CREATE ASSERTION    SALARY_CONSTRAINT
CHECK  ( NOT EXISTS
          ( SELECT  *
            FROM    EMPLOYEE E, EMPLOYEE M,
                    DEPARTMENT D
            WHERE   E.Salary > M.Salary
                    AND E.Dno = D.Dnumber
                    AND D.Mgr_ssn = M.Ssn) ) ;
```

- **CHECK**

Kan bij:
attribuut (al gezien)
CREATE DOMAIN
Enkel gecontroleerd bij toevoegen/wijzigzen van tupels

CHECK VS ASSERTION
Check = efficiënter, maar minder algemeen

Actie d.m.v. triggers

Event + voorwaarde + actie
Syntax

```
CREATE TRIGGER <name>
{ BEFORE | AFTER } <event> ON <table>
FOR EACH ROW
WHEN (<cond>)
    <action>
```

Link met (E)ER = implementatie van een afgeleid attribuut


```
CREATE TRIGGER Total_sal1
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.Dno IS NOT NULL)
      UPDATE DEPARTMENT
      SET Total_sal = Total_sal + NEW.Salary
      WHERE Dno = NEW.Dno
```

Elementen van triggers

Timing: before / after / instead of
Actie kan verwijzen naar oude of nieuwe toestand van de gegevensbank
Conditie wordt gespecificeerd in WHEN clausule

Relationele calculus

Algemeen

Calculustalen

- Steunen op relationele calculus
- Declaratief **(WAT)**: formele beschrijving van gewenste info adhv predikatenlogica
- 2 soorten:
  - Tupelcalculus
  - Domeincalculus (niet gezien)

Relationele algebra

Beschrijft operaties om tot resultaat te komen

Relationele calculus

Beschrijft condities waaraan resultaat moet voldoen

== procudereel == declaratief

Omschrijving gegeven in predikatenlogica

Ter herinnering: predikatenlogica

Gebruikt variabelen om een bewering over objecten te maken

Eerste orde predikatenlogica: na ∃, ∀ (= quantoren, zie verder) geen complete verzamelingen, enkel variabelen

Propositie	Operatoren	Samengestelde beweringen
Bewering die juist of fout is	Ontkenning ¬	Één of meer symbolen
Aangegeven dmv 1 symbool (p,q,…)	Conjunctie ∧	operatoren
	Disjunctie ∨	
	Implicatie ⇒	
	Equivalentie ⟶	

Tupel relationele calculus (tupelcalculus)

Algemeen

Algemene basisvorm van query:

$$\{ t_1.A1, t_2.A2, \ldots, t_n.A_n \mid COND(t_1, t_2, \ldots t_n, t_{n+1}, \ldots, t_{n+m}) \}$$

t_i Tupelvariabele (mag ook allemaal dezelfde t zijn)
Opm

A_i Attribuutnaam, horende bij relatie t_i

COND voorwaarde (of predikaat of "well formed formula - WFF)

Opmerking: t_i links van "|" = alle variabelen die vrij voorkomen (niet gebonden door ∃ of ∀) in de COND

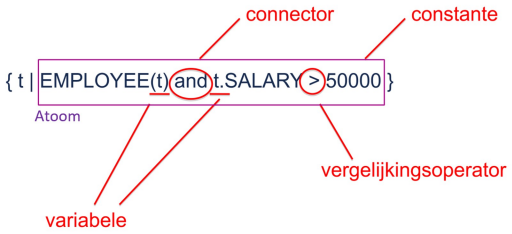
COND bestaat uit:

Atomen + logische connectoren + quantificaties

Atoom heeft waarde true / false en is van de vorm:

- $R(t_i)$ R: relatienaam, t_i: tupelvariable
- $t_i.A\ \theta\ t_j.B$ $\theta \in \{ <,>,=, \leq, \geq, \neq \}$ en A attribuut van t_i, B attribuut van t_j
- $t_i.A\ \theta\ c$ of $c\ \theta\ t_i.B$ $\theta \in \{ <,>,=, \leq, \geq, \neq \}$ en c een constante

Vb.



Gebonden en vrije variabelen

Gebonden = een tupelvariabele is gequantificeerd (komt voor met ∃ of ∀)

F_1: d.DNAME = 'Research'

F_2: (∃ t) (d.DNUMBER = t.DNO)

F_3: (∀ d) (d.MGSSN = '333445555')

d is vrij in F_1 en in F_2

d is gebonden met ∀ in F_3

t is gebonden met ∃ in F_2

Quantoren

Existentiële (∃)

(∃t)(formule(t))

- er bestaat een tupel t die aan de conditie voldoet
- bv. (∃t)(EMPLOYEE(t) and t.fname= 'John')

- Geef naam en adres van alle werknemers die voor het 'Research' departement werken

```
{ t.Fname, t.Lname, t.Address | EMPLOYEE(t)
                                AND (∃d) (DEPARTMENT(d)
                                      AND d.Dname = 'Research'
                                      AND d.Number = t.Dno) }
```

existentiële quantor: moet waar zijn voor tenminste één tupel

we vragen werknemers op waarvoor er een gerelateerd tupel in de departement tabel bestaat met als naam 'Research'

Universele (∀)

(∀t)(formule(t))

- alle tupels *in het universum* voldoen aan de conditie

- Logische implicatie

$(\forall x)\ (boot(x) \Rightarrow (x.color = 'Rood'))$

'als x een boot is, is het een rode boot

analoog aan

$(\forall x)\ (\neg Boot(x)\ \vee x.color = 'Rood')$

'ofwel is x geen boot ofwel is x een rode boot'

Tweede vorm = **correct gebruik** in relationele calculus:

(∀x) (NOT (RELATION(x)) OR <test on x>)

- *Als* x een tupel in RELATION is, *dan* moet aan test worden voldaan.
- Predikaat geeft alleen true als voor alle tupels

we vragen werknemers op waarvoor er een gerelateerd tupel in de departement tabel bestaat met als naam 'Research'

(∀x) (NOT (RELATION(x)) OR <test on x>)

- *Als* x een tupel in RELATION is, *dan* moet aan test worden voldaan.
- Predikaat geeft alleen true als voor alle tupels in RELATION aan test wordt voldaan.

vb. (werknemers die aan alle projecten werken)

```
{ e.Lname, e.Fname |
 EMPLOYEE(e)
 AND
 ( (∀x) ( NOT (PROJECT(x))
          OR
          ( (∃w) ( WORKS_ON(w)
                   AND w.Essn = e.Ssn
                   AND x.Pnumber = w.Pno ) ) ) ) }
```

In woorden: *als* x een project is, *dan* moet er een overeenkomstig tupel in de WORKS_ON tabel bestaan.

Quantoren in SQL

- ∃ EXISTS
- ∀
  - bestaat niet
  - gebruik not (∃ x): NOT EXISTS

Voorbeeld analogie:
$\forall\, a: red(a)$ ⟶ alle a rood
$\neg(\exists a)$: $\neg\, red(a)$ ⟶ er bestaat geen a die niet rood is (=alle a rood)

SAMENVATTING LES 7

woensdag 13 mei 2020 13:21

Functionele afhankelijkheden en normalisatie 1

Een gegevensbank ontwerpen: informele richtlijnen

Ontwerp van relationele gegevensbanken

- 2 benaderingen:
  1) Via hoogniveau modellering (**top-down**)
     - Werkelijkheid analyseren
     - ER-schema bouwen
     - ER-schema omzetten naar relationeel gegevensbankschema

  2) Meteen relationel gegevensschema bouwen (**bottom-up**)
     - Informele richtlijnen
     - Technieken , algoritmes

- Belangrijke doelen
  - Informatie bewaren
  - Minimale redundantie, maximale performantie, …

  => **NORMALISEREN**

Informele richtlijnen

1) Ontwerp een relatieschema zo dat zijn betekenis makkelijk verklaard kan worden
   - Betekenis van een relatie moet duidelijk zijn
   - Betekenis van attributen moeten duidelijk zijn

2) Ontwerp een relatieschema zo dat redundantie vermeden wordt en geen toevoeg-, weglaat- of wijziging-anomalieën kunnen voorkomen

Toevoeg-anomalieën

EMP_DEPT

Ename	SSN	Bdate	Address	Dnumber	Dname	Dmgr

- Nieuwe werknemer toevoegen
  - Nieuwe gegevens moeten consistent zijn met bestaande departement gegevens
  - Nul waarden als er geen dept info voorhanden is
- Hoe een departement toevoegen dat nog geen wernemers heeft?

Weglaat-anomalieën

EMP_DEPT

Ename	SSN	Bdate	Address	Dnumber	Dname	Dmgr

- Laatste werknemer uit een departement weg -> alle info over dat departement ook weg!
- Oplossing is bv: allemaal null waarden invullen bij employee

Wijziging-anomalieën

EMP_DEPT

Ename	SSN	Bdate	Address	Dnumber	Dname	Dmgr

- Bv. Manager van dept 5 wijzigen -> wijziging moet in alle tupels van EMP_DEPT met Dnumber = 5 gebueren, anders wordt gegevensbank inconsistent
- Minder cruciaal, redelijk makkelijk in update query

3) Vermijd zoveel mogelijk attributen waarvan de waarde nul kunnen zijn
   - Nul = niet van toepassing / onbekend / nog niet geregistreerd

4) Ontwerp relatieschema's zo dat ze na een equi-join op gerelateerde attributen geen onechte tupels opleveren
   - Vb. EMP_LOCS en EMP_PROJ => na natuurlijke join worden onechte tupels ingevoerd

Een voorbeeld van normalisatie

- "Kaartenbak"-voorbeeld
  - Context: "open universiteit"

- Kaarten met gegevens over:
  - StudentNr, Naam, Cursus
  - Cursus =

- Code, Titel, InschrijvingsDatum, StudiecentrumCode, StudiecentrumNaam

- Een student kan meerdere cursussen volgen
  - Cursus is een meerwaardig samengesteld attribuut
    - Niet toegelaten in relationeel model!
    - Weghalen van meerwaardige / samengestelde attributen
      => EERSTE NORMAALVORM

- Bekijk gegevens: welke kunnen aanleiding geven tot anomalieën?
  - Waar is er redundantie?
  - M.a.w. welke gegevens volgen uit andere?

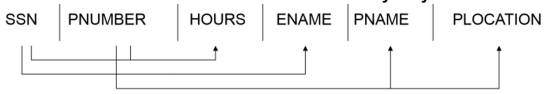
- Identificatie van afhankelijkheden
  - StudiecentrumNaam volgt uit StudiecentrumCode

- Herwerken van relatieschema's
  => TWEEDE EN DERDE NORMAALVORM

Functionele afhankelijkheden

Doel van functionele afhankelijkheden

- Expliciet beschrijven van bepaalde restricties waaraan tupels in de relatie steeds moeten voldoen
- Notatie in schema's via pijlen *(Geen verwijssleutels!!)*
  - Aanduiden welke attributen afhankelijk zijn van andere attributen



Definitie functionele afhankelijkheden

Zij gegeven een relatieschema R = {A1, A2, … , An} en attributenverzamelingen X en Y
We zeggen dat Y functioneelafhankelijk is van X, genoteerd X → Y, als en slechts als:
1) X ≠ ∅
2) ∀ mogelijke extensies r van R:
   t1, t2 ∈ r en t1[X] = t2[X] ⇔ t1[Y] = t2[Y]

Gewone woorden: als twee tupels hetzelfde ssn hebben, gaan ze ook dezelfde ename hebben => ename is functioneel afhankelijk van ssn

Afleidingsregels voor functionele afhankelijkheden

- Een verzameling functionele afhankelijkheden van R kan nieuwe functionele afhankelijkheden impliceren
  - Bv. A → B en B → C impliceert A → C
  - Notatie: {A → B, B → C} |= A → C

- Afleidingsregels voor functionele afhankelijkheden
  - Laten toe alle functionele afhankelijkheden te vinden die door een verzameling F van functionele afhankelijkheden geïmpliceerd worden

- Sluiting van F, genoteerd F+
  - Is de verzameling van alle f.a. die door F geïmpliceerd worden

Basis afleidingsregels
FD1. reflexiviteitsregel: $Y \subseteq X \Longrightarrow X \rightarrow Y$
FD2. uitbreidingsregel: {X → Y} |= XZ → YZ
FD3. transitiviteitsregel: {X → Y, Y → Z} |= X → Z

- Basisregels zijn correct en volledig
  - Correct (sound)
    - Alle afhankelijkheden die afgeleid kunnen worden, worden ook werkelijk geïmpliceerd
  - Volledig (complete)
    - Alle afhankelijkheden die geimpliceerd worden, kunnen ook met de regels afgeleid worden

Extra afleidingsregels
FD4. decompositieregel {X → YZ} |= X → Y
FD5. verenigingsregel {X → Y, X → Z} |= {X → YZ}
FD6. pseudo-transitiviteitsregel {X → Y, WY → Z} | WX → Z

Hoe alle functionele afhankelijkheden in een relatieschema vinden?

- Uit betekenis van relaties en attributen: → verzameling afhankelijkheden F

- Onvermijdelijk: afhankelijkheden volgen uit betekenis
- Via afleidingsregels worden alle functionele afhankelijkheden bekomen die uit F volgen: F+
- Hoe:
  - Voor elke verzameling attributen X, die links in een functionele afhankelijkheid van F voorkomt, bepaal X_F+
- X_F+ ("sluiting van X t.o.v. F"):
  - = verzameling van alle attributen die functioneel bepaald zijn door X
- Algoritme:
  - Voor elke X → Y ∈ F, bepaal X_F+ met de afleidingsregels

Algorimte: bepaal X_F+, sluiting van X onder F

X_F+ := X
Herhaal
  oldX$_F$+ := X_F+
  Voor elke Y → Z **in** F:
    Als Y ⊆ X_F+ **dan** X_F+ := X_F+ U Z
Tot (oldX$_F$+ = X_F+)

G = {Ssn → {Ename, Bdate, Address, Dnumber},
Dnumber → {Dname, Dmgr_ssn}}

- $\{Ssn\}^+$ = {Ssn, Ename, Bdate, Address, Dnumber, Dname, Dmgr_ssn}
- $\{Dnumber\}^+$ = {Dnumber, Dname, Dmgr_ssn}

Overdekking, equivalentie

- Zij E en F verzamelingen f.a. van R

- E wordt **overdekt** door F a.s.a. E ⊆ F+
  - Alle f.a. in E volgen uit F via afleidingsregels
  - Automatisch ook E+ ⊆ F+

- E en F zijn **equivalent** a.s.a. E+ = F+
  - E overdekt F en F overdekt E
  - E+ ⊆ F+ en F+ ⊆ E+ ⟹ E+ = F+

- Nagaan of F E overdekt:
  - Voor alle X → Y in E: bepaal X_F+ en controleer of Y ⊆ X_F+

Overdekt F de verzameling E?
- E = {A → BC, D → AE}
- F = {A → B, AB → C, D → AC, D → E}

- voor alle X → Y in E:
  - bepaal X_F^+ en controleer of Y ⊆ X_F^+

- A → BC
  - A_F^+ = {A, B, C}
  - BC ⊆ {A, B, C}
- D → AE
  - D_F^+ = {D, A, C, E, B}
  - AE ⊆ {D, A, C, E, B}

Minimale verzameling functionele afhankelijkheden

- Een verzameling f.a. is minimaal a.s.a. voldoet aan
  - Rechterlid van elke afhankelijkheid in F bestaat uit 1 attribuut
  - Voor geen X → A en Z ⊂ X is (F \ { X → A }) U { Z → A } equivalent met F
  - Voor geen X → A van F is F \ {X → A} equivalent met F

- Dus: F bevat geen redundante afhankelijkheden, en de f.a.'s geen redundante attributen

- E is een minimale overdekking van F a.s.a. E F overdekt en minimaal is

Algoritme minimale overdekking

G := F
voor elke $X \rightarrow A_1A_2...A_n$ **in** G (met n > 1) **:**
  $G := G \setminus \{X \rightarrow A_1A_2...A_n\} \cup \{X \rightarrow A_1, ..., X \rightarrow A_n\}$

voor elke X → A **in** G **:**
  voor elke B ∈ X **:**
    X' := X \ {B}
    G' := G \ {X → A} ∪ {X' → A}
    als $B \in X'_{G'^+}$ **dan** G := G'

voor elke X → A **in** G **:**
  G' := G \ {X → A}
  als $A \in X_{G'^+}$ **dan** G := G'

Normaalvormen

- Afhankelijkheden in een relatieschema kunnen oorzaak zijn van
  - Redundantie van gegevens in een extensie
  - Problemen bij het onderhoud
- Daarom:
  - Indeling van relatieschema's in groepen op grond van soorten afhankelijkheden: NORMAALVORMEN
  - Hoe hoger de normaalvorlm, des te minder afhankelijkheden zich kunnen voordoen
  -
- Een **normaalvorm** legt bepaalde eisen op aan relaties

- **Normalisatie**

- = relatie in een bepaalde normaalvorm brengen
- Methode: decompositie
  - Relatieschema's die niet voldoen aan de eisen opdelen in kleinere relatieschema's die wel voldoen

- Een goede **decompositie** van het schema
  - Bevat dezelfde informatie
  - Bevat geen onnodige relaties
  - Is efficiënt te onderhouden

Decompositie

- Notatie: voor een relatie R noteren we:
  - Het relatieschema S_R
  - De attributenverzameling van S_R: U_R
  - Een verzameling afhankelijkheden van S_R: F_R
  - $S_R = <U_R, F_R>$

- **Definitie**:

Een verzameling relatieschema's
$$\delta(S_R) = \{S_{R1}, S_{R2}, ..., S_{Rk}\} \qquad (k > 1)$$
is een *decompositie* van S_R a.s.a. $U_R = U_{R1} \cup U_{R2} \cup ... \cup U_{Rk}$

Normaalvormen

- Opeenvolgende normaalvormen
  - 1NF
  - 2NF
  - 3NF
  - BCNF (Boyce-Codd normaalvorm)
  - 4NF
  - 5NF

  => elke normaalvorm is een speciaal geval van de vorige

Enkele definities

- **Supersleutel**: K is een supersleutel voor een relatie R met schema $S_R<U, F>$ a.s.a. $K \subseteq U$ en $K \rightarrow U \in F+$
  - Een supersleutel determineert elk attribuut in R

- **Kandidaatsleutel**: K is een sleutel of kandidaatsleutel voor een relatie R met schema $S_R<U, F>$ a.s.a. K een supersleutel is voor R en er geen enkele echte deelverzameling van K een supersleutel is voor R

- **Sleutelattribuut**: een attribuut A is een sleutelattribuut voor een relatie R met schema $S_R<U, F>$ a.s.a. er bestaat een sleutel K voor R waarvoor geldt $A \in K$

Nulde normaalvorm
Een relatie schema $S_R<U, F>$ is in nulde normaalvorm:
Er zijn geen voorwaarden opgelegd aan de attributen (m.a.w. ze mogen samengesteld of meerwaardig zijn)

Eerste normaalvorm

Definitie
Een relatie schema $S_R<U, F>$ is in eerste normaalvorm a.s.a. het domein van elk attribuut van U enkelvoudig is.

- M.a.w. geen verzamelingen of tupels als waarde van een attribuut toegelaten
- Nieuwe gegevensbank bevat zelfde informatie als de oorspronkelijke
- Merk op: er is geen enkele eis gesteld aan de verzameling functionele afhankelijkheden F

Relatie in 1NF brengen
- Samengesteld attribuut in meerdere attributen opsplitsen
- Voor meerwaardig attribuut meerdere tuples voorzien of nieuwe relatie creëren met zelfde sleutel

Tweede normaalvorm

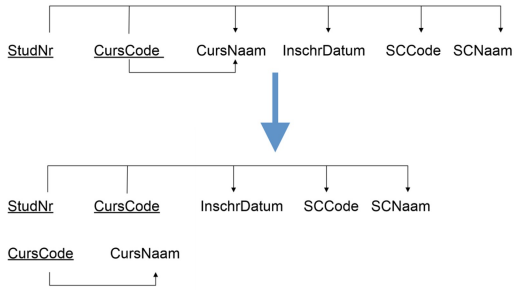
Definitie: partieel functioneel afhankelijk
Als $X \rightarrow Y$, dan zeggen we dat Y partieel functioneel afhankelijk is van X indien er een $Z \subset X$ bestaat zodat $Z \rightarrow Y$;
Anders noemen we Y volledig functioneel afhankelijk van X.

Definitie: tweede normaalvorm
Een relatie schema $S_R<U, F>$ is in tweede normaalvorm a.s.a voor geen enkel niet-sleutelattribuut A van U geldt dat A partieel functioneel afhankelijk is van een kandidaatsleutel van R

- M.a.w.: voor elk niet-sleutel-attribuut moet de hele primaire sleutel nodig zijn om het te determineren (en dit voor elke mogelijke keuze van primaire sleutel)

Methode 1NF → 2NF

- Gegeven $S_{R1} = < U_{R1}, F_{R1} >$ in 1NF
- Voor elk attribuut A dat partieel functioneel afhankelijk is van een kandidaatsleutel K:
  - Zoek een subset K' in K waarvan A volledig functioneel afhankelijk is
  - Elimineer A uit U_{R1}
  - Maak een nieuw relatieschema S_{R2} met attributenverzameling $U_{R2} = K' \cup A$



Derde normaalvorm

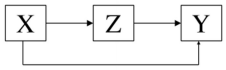
Definitie: triviale functionele afhankelijkheid

Een functionele afhankelijkheid X → Y is een triviale functionele afhankelijkheid a.s.a. $Y \subseteq X$

Definitie: transitieve functionele afhankelijkheid

Een functionele afhankelijkheid X → Y is een transitieve functionele afhankelijkhheid a.s.a. er een Z bestaat zodat volgende 3 voorwaarden voldaan zijn

1) Z is volledig en niet-triviaal functioneel afhankelijk van X
2) Z is geen (echte of onechte) deelverzamelijng van een kandidaatsleutel
3) Y is niet-triviaal functioneel afhankelijk van Z



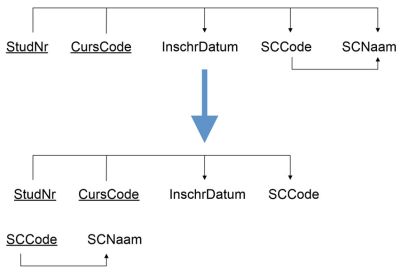
We zeggen dat Y transitief functioneel afhankelijk is van X via Z

Definitie: derde normaalvorm

Een 2NF-relatieschema $S_R<U, F>$ is in derde normaalvorm a.s.a voor geen enkel niet-sleutelattribuut A van U geldt:
A is transitief functioneel afhankelijk van een of andere kandidaatsleutel van de relatie R

Methode 2NF → 3NF

- Gegeven $S_{R1} = < U_{R1}, F_{R1} >$ in 2NF
- Voor elk niet-sleutelattribuut A dat transitief functioneel afhankelijk is van een kandidaatsleutel via Z:
  - Elimineer A uit U_{R1}
  - Maak een nieuw relatieschema S_{R2} met attributenverzameling $U_{R2} = Z \cup A$



SAMENVATTING LES 8

woensdag 13 mei 2020 19:01

Normalisatie 2

Boyce-Codd normaalvorm

- Bij 3NF zijn binnen sleutels nog functionele afhankelijkheden toegestaan
- Deze wegwerken → BCNF

Definitie BCNF

Een 3NF relatieschema S_R<U, F> is in Boyce-Codd normaalvorm a.s.a voor geen enkel sleutelattribuut A van U geldt:
A is partieel of transitief functioneel afhankelijk van een of andere kandidaatsleutel van R

Equivalente definitie BCNF

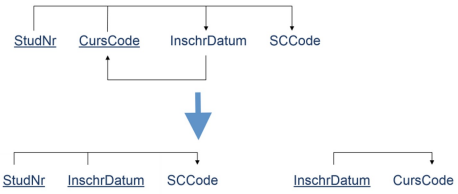
Een 1NF relatieschema S_R<U, F> is in Boyce-Codd normaalvorm a.s.a voor elke niet-triviale functionele afhankelijkheid X → A in F+ geldt dat X een supersleutel is

Methode 3NF → BCNF

- Gegeven S_{R1} = < U_{R1}, F_{R1} > in 3NF
- Voor elk sleutelattribuut A dat transitief functioneel afhankelijk is van een kandidaatsleutel via Z:
  - Elimineer A uit U_{R1}
  - Maak een nieuw relatieschema S_{R2} met attributenverzameling U_{R2} = Z U A
- Zoals 2NF → 3NF, maar voor sleutel attributen

BCNF ideaal m.b.t. functionele afhankelijkheden

- **JA**
  - Alle ongewenste functionele afhankelijkheden (patrieel en transitief, voor alle attributen) zijn weg
- **HELAAS**
  - Niet steeds bereikbaar zonder andere problemen te creëren
  - Sommige f.a. worden moeilijker te controleren



Redundantie is verwijderd;
maar: de f.a. StudNr, CursCode → InschrDatum, SCCode
wordt moeilijker te controleren!

Decompositie: algoritmen en eigenschappen

Benadering => meteen relationeel gegevensschema bouwen (**bottom-up**)

Algoritmen en eigenschappen

Vertrek van 1 universele relatie R
- Met schema S_R = < U_R, F_R >

Gewenst:
- Decompositie $\delta(S_R) = \{S_{R1}, S_{R2}, ..., S_{RK}\}$ (met k > 1) zodat
  - Elke R_i in BCNF of 3NF is
  - Alle informatie behouden blijft
  - Alle functionele afhankelijkheden bewaard blijven
- Laatste 2 voorwaarden zijn niet zo evident!!

Ideale decompositie?
1) Alle relaties in BCNF
2) Verliesloze decompositie
3) Afhankelijkheden bewaren

=> Helaas…You can't have it all! Slechts 2 uit 3 altijd haalbaar

Bewaren van afhankelijkheden

Functionele afhankelijkheden zijn intrarelationele restricties

- X → Y : X en Y zijn attributenverzamelingen van dezelfde relatie

Bij decompositie geven deze afhankelijkheden aanleiding tot:
- **Intrarelationele** afhankelijkheden in R_i: $\pi_{U\ Ri}(F_{R0})$
  - Projectie van F_{R0} op U_{Ri}:
    Alle X → Y in F+ waarvoor X U Y ⊆ U_{Ri}
- **Interrelationele** afhankelijkheden
  - Deze zijn moeilijker te controleren door DBMS:
    Join nodig om na te gaan of nieuw toegevoegde tupels voldoen

Een decompositie $\delta(S_R) = \{S_{R1}, S_{R2}, \ldots, S_{Rk}\}$ van R bewaart de afhankelijkheden van R indien
$(\pi_{UR1}(F_R) \cup \ldots \cup \pi_{URk}(F_R))+ = F_R+$

1) Alle relaties in 3NF
2) ~~Verliesloze decompositie~~
3) Afhankelijkheden bewaren

Algoritme voor afhankelijkheden-bewarende decompositie naar 3NF
1) Zoek een minimale overdekking G voor F
2) **Voor elke** X die ergens G voorkomt als de linkerzijde van een functionele afhankelijkheid:
   Maak een schema { X U A_1 U A_2 U … U A_m } (met primaire sleutel X)
   Waarbij X → A_1,…, X → A_m alle afhankelijkheden in G zijn met X als linkerlid
3) Plaats alle overblijvende attributen in een enkel relatieschema
   => kan tot informatieverlies leiden

Model#	Serial#	Price	Color	Name	Year

Dependencies
{M, S} → {P, C}
{S} → {Y}
{M} → {N}
{N,Y} → {P}

Minimal cover
{M, S} → {C}
{S} → {Y}
{M} → {N}
{N,Y} → {P}

Model#	Serial#	Color

Serial#	Year

Model#	Name

Name	Year	Price

Behoud van informatie
- Informatie blijft behouden a.s.a. oorspronkelijke relatie volledig gereconstrueerd kan worden uit decompositie

- Voorbeeld: EMP_LOCS

Ename	Plocation

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation

  - Oorspronkelijke relatie niet reconstrueerbaar: join levert onechte tupels op
  - Dus geen volledig behoud van informatie

Let op: behoud van informatie impliceert dus ook geen nieuwe tupels introduceren!!

Verliesloze decomposities
Een decompositie $\delta(S_R) = \{S_{R1}, S_{R2}, \ldots, S_{Rk}\}$ is verliesloos a.s.a. voor elke toegestane extensie r van R (d.w.z. een extensie die aan FR voldoet) geldt:
$\pi_{UR1}(r) * \ldots * \pi_{URk}(r) = r$

- M.a.w. natural join van relaties in decompositie levert geen onecthe tupels op (lossless of nonadditive join property)
- Testen of een decompositie verliesloos is d.m.v. het **Chase** algoritme

Chase-algoritme
- Maakt gebruik van een matrix
  - Kolommen = alle attributen A_j van originele relatie R
  - Een rij per relatie R_i in de decompositie
- Voor elke rij i:
  - Vul a_j in op positie j als A_j in schema S_{Ri} voorkomt; anders b_{ij}
  - De a's stellen de aanwezige waarden voor
- Redenering
  - Als X → Y, en de waarden voor X zijn gelijk in 2 rijen, dan ook die voor Y

- Dit aanduiden in matrix
- Doorgaan tot geen verandering meer optreedt (of rij a's)
- Rij vol a's ⇔ verliesloos

Niet-additieve Join Test voor Binaire Decomposities

$\delta(S_R) = \{S_{R1}, S_{R2}\}$ is verliesloos t.o.v. F a.s.a.

$(U_{R1} \cap U_{R2}) \rightarrow (U_{R1} \setminus U_{R2}) \in F+$

Of

$(U_{R1} \cap U_{R2}) \rightarrow (U_{R2} \setminus U_{R1}) \in F+$

- Eenvoudiger dan Chase algoritme
- Enkel toepasselijk op splitsing in 2 schema's
- Maar ook wanneer recursief toegepast!

Verliesloze decompostie naar BCNF

1) Alle relaties in BCNF
2) Verliesloze decompositie
3) ~~Afhankelijkheden bewaren~~

Algoritme verliesloze decompositie naar BCNF

δ := {SR}

Zolang er een relatieschema SRi in δ niet in BCNF is:

1) Zoek een functionele afhankelijkheid X → Y binnen SRi die volgens BCNF niet toegelaten is
2) Vervang SRi in δ door twee schema's SRi \ Y en X U Y:

δ := δ \ {SRi} U {SRi \ Y, X U Y }

=> afhankelijkheden kunnen nu verloren gaan

Verliesloze en afhankelijkhedenbewarende decompositie naar 3NF

1) Alle relaties in 3NF
2) Verliesloze decompositie
3) Afhankelijkheden bewaren

Algoritme verliesloze en afhankelijkhedenbewarende decompositie naar 3NF

1) Zoek een minimale overdekking G voor F
2) **Voor elke** X die ergens G voorkomt als de linkerzijde van een functionele afhankelijkheid:

   Maak een schema { X U A_1 U A_2 U … U A_m } (met primaire sleutel X)

   Waarbij X → A_1,…, X → A_m alle afhankelijkheden in G zijn met X als linkerlid
3) Als geen van de relatieschema's een sleutel van R bevat

   Dan maak een extra relatieschema dat attributen bevat die een sleutel voor R vormen

Emp_ssn	Pno	Esal	Ephone	Dno	Pname	Plocation

Afhankelijkheden

{Emp_ssn} → {Esal,Ephone,Dno}

{Pno} → {Pname,Plocation}

{Emp_ssn, Pno} → {Esal,Ephone,Dno,Pname,Plocation}

1. Minimale overdekking

   Emp_ssn → Esal

   Emp_ssn → Ephone

   Emp_ssn → Dno

   Pno → Pname

   Pno → Plocation

2.

Emp_ssn	Esal	Ephone	Dno

Pno	Pname	Plocation

3.

Emp_ssn	Pno

Algoritme voor vinden van een sleutel

Input: universele relatie R en verzamelng functionele afhankelijkheden F

1) K:= R
2) Voor elk attribuut A in K

   Bereken (K-A)+ t.o.v. F

   Als (K-A)+ allte attributen van R bevat, dan K:= K-{A}

Nulwaarden kunnen problemen geven

- Effect van nulwaarden op aggregaatfuncties?

- Ook problematisch voor reconstructie van originele relatie uit decompositie

- Nulwaarden in verwijssleutels
- Bv. 2 nieuwe werknemers, nog geen dept. → Dnum = nul
- EMPLOYEE*DEPARTMENT vermeldt die werknemers niet
- Worden "**dangling tuples**" genoemd
- Deze gaan verloren bij inwendige join
- Uitwendige join biedt oplossing

Nadelen van decompositiemethode

- Functionele afhankelijkheden moeten vooraf bekend zijn
  - Kan moeilijk zijn bij grote gegevensbanken met veel attributen
- De algoritmen zijn niet deterministisch
  - Minimale overdekking van F: meerdere oplossingen
    Decompositie is afhankelijk van volgorde waarin f.a. bekeken wordt
- Na decompositie (dure) joins nodig om originele informatie terug te vinden
  - Kan performantie nadelig beïnvloeden

Besluit

- Methode niet blindelings toepassen
- Combinatie van normalisatie $_{en}$ ER-ontwerp is aangewezen
- Bv:
  - Begin met (E)ER-model
  - Beeld af op relationeel model
  - Zoek functionele afhankelijkheden
  - Controleer of verdere decompositie nodig/ wenselijk is

Vierde normaalvorm

Meerwaardige afhankelijkheden

- Meervoudige afhankelijkheden zijn een gevolg van 1NF

Course	Teacher	Text
Physics	Green Brown Black	Mechanics Thermodynamics
Math	White	Algebra Geometry

- Betekenis: elke docent gebruikt elke tekst (onafhankelijkheid van Teacher en Text)

- X bepaalt eenduidig een verzameling Y's ("X multidetermineert Y")
- Welke Y's bij X horen hangt niet af van andere attributen

Definitie: meerwaardige afhankelijkheden

Een attributenverzameling Y van een Relatie R is meerwaardig afhankelijk van een attributenverzameling X van R, genoteerd X ->> Y a.s.a.
Voor elke toegestane extensie r van R geldt:
Als er 2 tupels t_1 en t_2 in r zijn zodat $t_1[X] = t_2[X]$,
Dan bestaan er twee tupels t_3 en t_4 in r zodat
$t_3[X] = t_4[X] = t_1[X] = t_2[X]$ en
$t_3[Y] = t_1[Y]$ en $t_4[Y] = t_2[Y]$ en
$t_3[U_R \backslash X \backslash Y] = t_2[U_R \backslash X \backslash Y]$ en $t_4[U_R \backslash X \backslash Y] = t_1[U_R \backslash X \backslash Y]$

gegeven een relatie R, X en Y zijn deelverzamelingen van attributen in R en Z = R – (X U Y)

X ->> Y
als er 2 tupels t_1 en t_2 in r zijn zoda $t_1[X]=t_2[X]$, dan bestaan er twee tupels t_3 en t_4 in r zodat

– $t_3[X] = t_4[X] = t_1[X] = t_2[X]$ en
– $t_3[Y] = t_1[Y]$ en $t_4[Y] = t_2[Y]$ en
– $t_3[Z] = t_2[Z]$ en $t_4[Z] = t_1[Z]$

	X	Y	Z
t_1	a	b_1	c_1
t_2	a	b_2	c_2
t_3	a	b_1	c_2
t_4	a	b_2	c_1

- Als X meerdere attribuutverzamelingen multidetermineert: combinatorische explosie!
- Vb: om <x1, {a1, a2}, {b1, b2, b3 }, {c1, c2, c3} > voor te stellen: 18 tupels nodig
- Daarom meerwaardige afhankelijkheden weren uit relaties → **4NF**

Defenitie: triviale meerwaardige afhankelijkheden

Een meerwaardige afhankelijkheid X->>Y in relatie R is een triviale meerwaaridge afhankelijkheid a.s.a.
$Y \subseteq X$ of $X \cup Y = U_R$

Vierde normaalvorm

Definitie vierde normaalvorm

Een relatieschema S_R = <U, F> is in vierde normaalvorm a.s.a.
Voor elke niet trivieale meerwaardige afhankelijkheid van de vorm X->>Y van F+ geldt:
X is een supersleutel van R

- Belang van 4NF: vermijden van redundantie en anomalieën
- X->>Y en X geen supersleutel: decompositie naar twee schema's met attributen U \ Y en X U Y

Course ->> Teacher en Course ->>Text

Course	Teacher	Text
Physics	Green	Mechanics
Physics	Green	Thermodynamics
Physics	Brown	Mechanics
Physics	Brown	Thermodynamics
Physics	Black	Mechanics
Physics	Black	Thermodynamics
Math	White	Algebra
Math	White	Geometry

⇨

Course	Teacher
Physics	Green
Physics	Brown
Physics	Black
Math	White

Course	Text
Physics	Mechanics
Physics	Thermodynamics
Math	Algebra
Math	Geometry

Vijfde normaalvorm

Join afhankelijkheid

Definitie: join afhankelijkheid

Een join-afhankelijkheid JD(U_1, …, U_n) in relatie R is een restrictie die aangeeft dat voor elke extensie r van R geldt:

Er is een verliesloze decompositie in relaties R_1, …, R_n (n>2) met attributen U_1, U_2, …, U_n;

M.a.w. $\pi_{U1}(r) * \pi_{U2}(r) * \pi_{Un}(r) = r$

- Voorbeeld: JD({Agent, Company}, {Company, Product}, {Agent, Product}
- Merk op! => wanneer n=2 spreken we van een meerwaardige afhankelijkheid ($U_1 \cap U_2$ ->> $U_1 \setminus U_2$)

Model	Maat	Kleur	
x	y		als dergelijke tupels voorkomen in een extensie van R...
x		z	
	y	z	
x	y	z	dan ook dit tupel

Dit is de join-afhankelijkheid
JD ({Model,Maat} , {Maat,Kleur} , {Model,Kleur})

Definitie: triviale join-afhankelijkheid

Een join-afhankelijkheid JD(U_1, …, U_n) in relatie R is een triviale join-afhankelijkheid a.s.a.
Een van de U_i gelijk aan de U_R

Model	Maat	Kleur	
x	y		als dergelijke tupels voorkomen in een extensie van R...
x		z	
x	y	z	
x	y	z	dan ook dit tupel

Vijfde normaalvorm

Definitie: vijfde normaalvorm

Een relatieschema S_R = <U, F> is in vijfde normaalvorm (project-join-normaalvorm) a.s.a.
Voor elke niet-triviale join-afhankelijkheid JD(U_1, …, U_n) van F+ geldt: U_1, …, U_n zijn allemaal supersleutels van R

Methode om relatieschema naar 5NF te normaliseren

Voor elke JD(δ) in F_R+ : splits R op volgens δ

Voorbeeld : T-shirts
U_R = { Model, Maat, Kleur }
JD ({Model,Maat} , {Maat,Kleur} , {Model,Kleur})
splits op in { Model, Maat }, { Maat, Kleur }, { Model, Kleur }

Samenvatting les 9

donderdag 14 mei 2020 11:46

Programma's verbinden met gegevensbanken

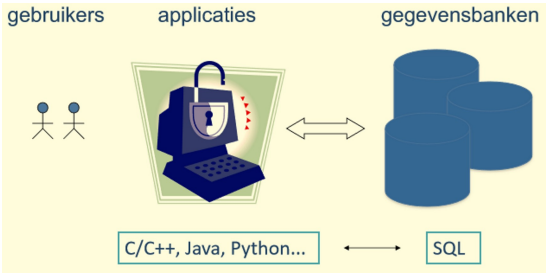
Benaderingen voor gegevensbankprogrammeren

Algemeen

Gebruik van SQL

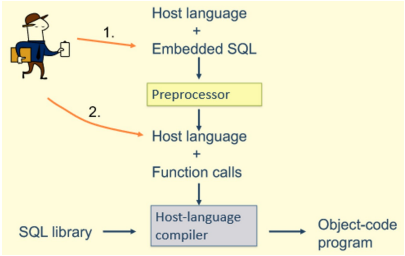
• Interactief	• Vanuit applicatie
Command line client lokale client met GUI: vb MySQL Workbench Web-based: vb phpMyAdmin Opdracht wordt aangeboden en direct uitgevoerd	SQL wordt gebruikt in een programma **== focus van les**

Programmeren met GBen **in 2-tierarchitectuur**
= software opgebouwd uit twee lagen



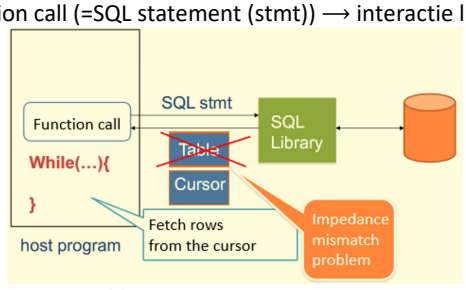
3 mogelijke **benaderingen** voor **interactie tussen applicaties en GBen**:

1. Inbedden van SQL opdrachten in programmacode
2. Gebruik van library met functies
   Functies verschaffen toegang tot GB en stellen resultaat beschikbaar aan een programma in een API
3. (Nieuwe programmeertaal ontwikkelen) = niet relevant



SQL/Host Language Interface

Function call (=SQL statement (stmt)) ⟶ interactie library met GB ⟶ geen tabel maar cursor meegeven en zo rows fetchen (tuper per tupel)



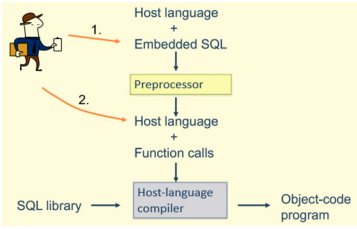
Waarom niet in één keer hele tabel? **Impedance mismatch problem**

- verschil in syntax tussen gegevensmodel en programmeertaal (CHAR ⟷ String, INTEGER ⟷ int,…)
  - Op te lossen met 'bindings' (apart per programmeertaal)
    = mapping van datastructuur van resultaat aan datastructuur van het programma
- representatie van tabelstructuur vaak onmogelijk als resultaat weer te geven (tupel per tupel is makkelijker)

Typische interactie tussen client en server model

Client opent verbinding:
Vereist adres van machine + evt. authorisatie (login & paswoord)
Interactie tussen programma en GB
Queries,updates,…
Programma sluit verbinding met gegevensbank

Ingebedde SQL (1.)



= embedded SQL ⟶ SQL-opdracht in programmacode ingebed:

- Pre-compiler verwerkt de SQL-opdracht voor compilatie
  SQL-opdracht wordt vervangen door declaraties en functie-oproepen in de programmeertaal
- Verschillende syntaxen

Verbinding met GB
Verbinding maken
CONNECT TO *<server name>* AS *<connection name>*
AUTHORIZATION *<username & password>*;
Wijzigen actieve connectie (slechts één actieve mogelijk)
SET CONNECT *<connection name>*;
Verbinding beëindigen
DISCONNECT *<connection name>*;

Shared variables (gemeenschappelijke variabelen == voordeel van ingebedde SQL)
SQL-opdrachten kunnen verwijzen naar programmavariabelen ⟶ manier om informatie uit te wisselen
Opmerking:
Dubbel punt ":" als prefix
Moeten gedeclareerd worden in een **SQL declaratiesectie**

Embedded SQL in C

- Declaratie van variabelen

```
EXEC SQL BEGIN DECLARE SECTION;
varchar dname[16], fname[16], lname[16];
char ssn[10], bdate[11], sex[2], minit[2];
float salary, raise;
int dno, dnum;
int SQLCODE ; char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;
```

- Vb.

```
loop = 1;
while (loop) {
   prompt("Enter a social security number: ", ssn);
   EXEC SQL
     select Fname, Minit, Lname, Address, Salary
     into :fname, :minit, :lname, :address, :salary
     from EMPLOYEE
     where SSN = :ssn;
   if (SQLCODE == 0)
        printf(fname, minit, lname, address, salary)
   else printf("social sec. numb. does not exist: ", ssn);

   prompt("More s.s.numbers (enter 1 for Yes, 0 for No): ",
          loop) ;
   }
```

Hier wordt resultaat uit GB gebonden aan eerdee gedeclareerde variabelen

Opmerking: bidirectionele verbinding
- Variabele waarde gegeven in hostprogramma kan naar verwezen worden (vb. :ssn)
- GB kan gegevens in variabelen steken die dan weer accessible zijn door hostprogramma

SQLJ in Java

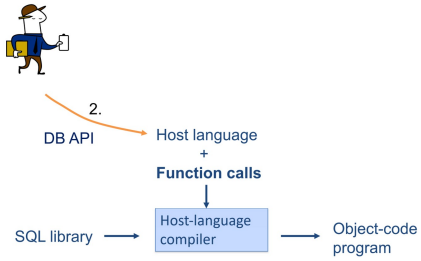
- SQLJ vertaler compilet de SQL commando's (omzetten in Java) d.m.v. JDBC interface
- Vereist:
  - Importeren van verschillende klassen (JDBC, IO,…)
  - Connecteren met gewenste GB (=getConnection())

Vb.

```
ssn = readEntry("Enter a social security number: ");
try {
    #sql{select Fname, Minit, Lname, Address, Salary
         into :fname, :minit, :lname, :address, :salary
         from EMPLOYEE where Ssn = :ssn};
}   catch (SQLException se) {
    System.out.println("s.s.number does not exist: "+ ssn);
    return ;
    }

System.out.println(fname + " " + minit + " " + lname + " "
                   + " " + salary)
```

Gegevensbanken via APIs (2.)

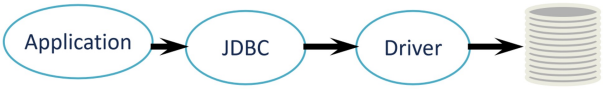


JDBC (= interface databasetoegang vanuit Java) als vb

Java en JDBC

- SQL statements dynamisch gemaakt en doorgegeven als strings
- JDBC gebruikt gestandardiseerde API voor toegang tot DBMS (veel DBMS leveranciers bieden JDBC aan)
- JDBC driver = implementatie van de functie-aanroepen die gespecificeerd zijn in de JDBC API

Architectuur JDBC:



Java code roept JDBC library aan ⟶ JDBC laadt driver ⟶ driver spreekt met GB

- **Werking**

1. Connectie maken: *getConnection(url,"username","password")*

```
  Connection con = DriverManager.getConnection
("jdbc:postgresql:employee", "postgres", "pwd");
```

2. Opdracht creëren: *Statement* object en gewenste string aanmaken

```
Statement stmt = con.createStatement();
 String creeer= "CREATE TABLE Department (" +
                "  Dname         Varchar(15) Not Null,"+
                "  Dnumber       Integer     Not Null,"+
                "  MgrSsn        Char(9),"+
                "  MgrStartDate DATE,"+
                " PRIMARY KEY (Dnumber),"+
                " UNIQUE (Dname)"+
                ")";
```

3. Opdracht uitvoeren: *executeQuery* of *executeUpdate*

```
try{
      stmt.executeUpdate (creeer);
      stmt.executeUpdate("COMMIT");
    }
    catch(SQLException sqle){
        stmt.executeUpdate("ROLLBACK");
        throw new FailureNoticeException(sqle.getMessage());
    }
stmt.close();
```

4. Resultaten ophalen: resultaat is van type *ResultSet*

```
ResultSet rs = stmt.executeQuery
    ("SELECT dname, dnumber, mgrssn, mgrstartdate " +
     " FROM Department" +
     " WHERE dnumber = " + number  );
if (!rs.next()) throw new FailureNoticeException
    ("Geen departement voldoet aan het zoekcriterium.");
Department d = new Department(rs.getString("dname"),
           rs.getString("dnumber"), rs.getString("mgrssn"),
           rs.getString("mgrstartdate"));

rs.close();
```

- **Prepared statement**

  Mogelijkheid tot concatenatie binnen strings ipv telkens nieuwe strings te maken

Opmerkingen	Code

'?' is placeholder voor later gevraagde waarde
 	
Verkregen ssn invullen in eerste vraagteken
 	
Zolang er rijen zijn eerste (1?) en tweede (2?) kolom uitprinten
	`Connection conn = DriverManager.getConnection`
	`("jdbc:oracle:oci8:" + dbacct + "/" + passwrd) ;`
	`String stmt1 =`
	`"select Lname, Salary from EMPLOYEE where Ssn = ?" ;`
	`PreparedStatement p = conn.prepareStatement(stmt1) ;`
	`ssn = readentry("Enter a Social Security Number: ") ;`
	`p.clearParameters() ;`
	`p.setString(1, ssn) ;`
	`ResultSet r = p.executeQuery() ;`
	`while (r.next()) {`
	`lname = r.getString(1) ;`
	`salary = r.getDouble(2) ;`
	`system.out.printline(lname + salary) ;`
	`}`

- **SQLJ vs JDBC**

	SQLJ	JDBC
SQL statements	statisch	Dynamisch
Strong typing	Yes	No
Type checking	Static	Runtime only
Syntax	Korter	API
Standard	ANSI	Sun
Portable	Yes	Yes
Object support	Yes	yes

SQLJ

```
#sql [ctx] {
  INSERT INTO DSN8710.EMP
    (EMPNO,  FIRSTNME,  MIDINIT,
   LASTNAME,  HIREDATE,  SALARY)
  VALUES
(:empno, :firstname, :midinit,
  :lastname, CURRENT DATE,
  :salary)
};
```

JDBC

```
stmt = conn.prepareStatement(
   "INSERT INTO DSN8710.EMP " +
   "(EMPNO, FIRSTNME, MIDINIT,
  LASTNAME, HIREDATE, SALARY) "
+ "VALUES (?, ?, ?, ?, CURRENT
   DATE, ?)");
stmt.setString(1, empno);
stmt.setString(2, firstname);
stmt.setString(3, midinit);
stmt.setString(4, lastname);
stmt.setBigDecimal(5, salary);
stmt.executeUpdate();
stmt.close();
```

Python

- **Python DB-API**
  - Gestandaardiseerde API
  - Verschillende implementaties beschikbaar (DBMS-specifiek, generiek (vb. ODBC = onze focus), …)

Pyodbc

(Open source dus voor veel besturingssystemen beschikbaar)
(gebruikt ODBC drivers dus kan met veel DBMSen communiceren)

- **Werking**

1. Connectie maken en cursor creëren

```
cnxn = pyodbc.connect('DRIVER={SQL Server};
      SERVER=localhost;DATABASE=testdb;UID=me;PWD=pass')
cursor = cnxn.cursor()
```

2. Gegevens opvragen

```
cursor.execute("select user_id, user_name from users")
row = cursor.fetchone()
if row:
    print row.user_name
```

```
cursor.execute("select user_id, user_name from users")
rows = cursor.fetchall()
for row in rows:
    print row.user_id, row.user_name
```

3. Data toevoegen

- Met ODBC kan je "?" gebruiken als parameter en de waarde daarna los meegeven
- Commit() nodig om wijzigingen door te voeren!

```
cursor.execute("insert into products(id, name) values
               (?, ?)", 'pyodbc', 'awesome library')
cnxn.commit()
```

- **Prepared statement?**

Analoog aan JDBC ⟶ %s ipv ? als placeholder

```
sql_update_query =
      """UPDATE Employee set Salary = %s where Id = %s"""
data_tuple = (12000, 1)
cursor.execute(sql_update_query, data_tuple)
connection.commit()
```

SQL injection

Ervoor zorgen dat de door de klant meegegeven waarden de query zo manipuleren dat je aan inloggegevens kan

The query will be:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND
Password='1' OR '1' = '1'
```

↳ altijd waar dus alle rijen in meegegeven (=ongewenst)

Te voorkomen met prepared statements? *"hangt ervan af, maar wel aangeraden om te gebruiken"*