Hoofdstuk 2

Het relationele model

Het **relationele model** werd in 1970 geïntroduceerd door de Amerikaanse wiskundige E. F. Codd. Dit model komt zo eenvoudig over dat het lange tijd is verguisd als zijnde niet-volwassen. Het eenvoudig concept, waarbij gegevens weergegeven worden in een verzameling gelijksoortige tabellen, is nochtans één van de belangrijkste voordelen van het relationele model t.o.v. het hiërarchische- en het netwerkmodel (vooral vroeger vormden deze laatste modellen de basis voor databanken.)

Bovendien heeft het relationele model een degelijke wiskundige basis en dat heeft een aantal voordelen bij het ontwerpen en onderhouden van relationele gegevensbanken. Sedert de introductie in 1970 is het model regelmatig herzien en aangepast. Het relationele model vormt momenteel dan ook de grondslag van de grootste groep DBMS-producten.

De student kan de bouwstenen van relationele gegevensbanken benoemen en bij een concreet voorbeeld aanduiden. Het gaat hier om begrippen als relatie/tabel, tupel/record, cartesisch product, attribuut, kenmerken van attributen in een relationele gegevensbank, domein van een attribuut, de betekenis van de null-waarde.
De student kan de kandidaat-sleutels, de primary key en de foreign key in een relationele databank bepalen. Hij kan voor een eenvoudige casus de relaties (tabellen), attributen en sleutels bepalen en het datastructuurdiagram tekenen. De student kan de begrippen redundantie, entiteit-integriteit, referentiële integriteit en domeinrestrictie uitleggen en toepassen op een concreet voorbeeld.
De student kent de operatie van de relationele algebra (selectie, projectie, join (natuurlijke join, equi-join, auto-join, outer-join (left-right), unie, doorsnede, verschil en kan dit toepassen om concrete casussen. Hij weet wat een view is en wat de bedoeling is van een view.

Het is belangrijk de theorie te begrijpen, maar probeer vooral de oefeningen vaak te doen tot je de redenering erachter perfect snapt. Kijk ook naar voorbeelden van databanken die je in je omgeving ziet.

2.1 Bouwstenen van relationele gegevensbanken

2.1.1 Relaties

Het relationele model is gebaseerd op een tak van de wiskunde, met name de verzamelingenleer en in het bijzonder de leer van de relaties.

Een **verzameling** is een ongeordende serie gelijksoortige elementen.

Een **relatie** op 2 of meer verzamelingen is een deelverzameling van de productverzameling van de betreffende verzamelingen.

Stel de volgende twee verzamelingen met de opgesomde elementen:

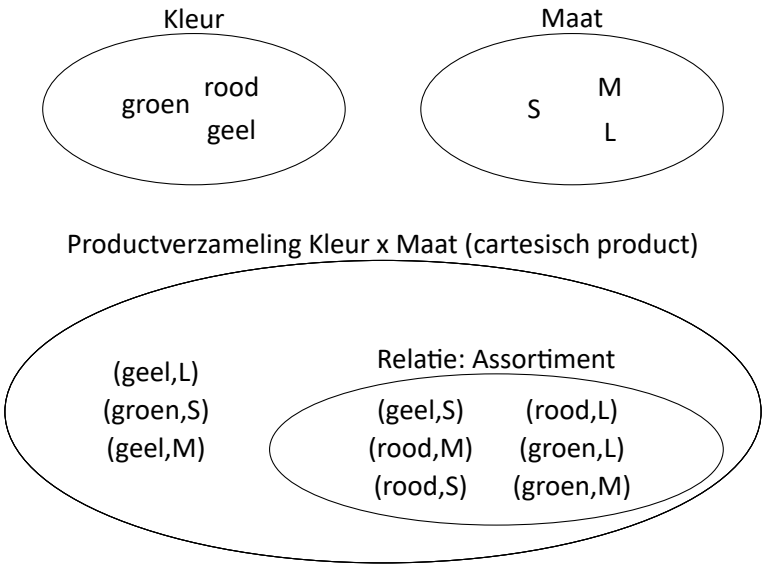
```
Kleur = {rood, geel, groen}
Maat = {small, medium, large}
```

De **productverzameling** `Kleur x Maat` bestaat uit 9 elementen:

```
{(rood, small), (rood, medium), (rood, large), (geel, small), (geel,
    medium), (geel, large), (groen, small), (groen, medium), (groen,
    large)}
```

Het aantal elementen v/d productverzameling is steeds het product van het aantal elementen van de vermenigvuldigde verzamelingen, hier 3 x 3 = 9.

Men noemt een dergelijk product van verzamelingen ook een **cartesisch product** (zie Figuur 2.1), het is natuurlijk totaal verschillend van een rekenkundig product van getallen.



Figuur 2.1: Illustratie cartesisch product.

Een voorbeeld van een relatie op de twee verzamelingen zou kunnen zijn:

```
{(rood, small), (rood, medium), (rood, large), (geel, small), (groen,
    medium), (groen, large)}
```

Als mogelijke betekenis zou deze relatie het assortiment jassen kunnen voorstellen welke een winkel verkoopt.

In een relationele databank wordt een relatie weergegeven d.m.v. een tabel.

De vijf elementen of tupels van de relatie Assortiment worden als volgt in tabelvorm voorgesteld:

Kleur	Maat
rood	S
rood	M
rood	L
geel	S
groen	M
groen	L

Elke rij uit de tabel representeert een tupel van de relatie en correspondeert bijgevolg met een **object** uit de werkelijkheid.

2.1.2 Attributen

In de theorie van relationele gegevensbanken draait dus alles om relaties, en in een relationele gegevensbank correspondeert met elke relatie een tabel, daardoor ziet een relationele gegevensbank er voor een gebruiker uit als een verzameling tabellen.

Een vereenvoudigde gegevensbank betreffende de personeelsadministratie van de Vives-hogeschool zou kunnen bestaan uit volgende 2 tabellen.

Employee

surname	name	id_employee	birthdate	sex	pay	id_supervisor	id_department
Acx	Johan	6541	15-01-1963	M	5222,62	7365	2
Desplenter	Marc	4379	19-02-1962	M	5202,88	7365	2
Ketels	Bavo	8167	12-04-1988	M	4602,88	7365	2
Haegeman	Wim	1234	31-12-1970	M	6718,40	7582	5
Hindryckx	Joris	7582	01-01-1960	M	8197,34	null	1
Beyls	Katrien	6741	null	V	4478,94	7365	2
De Langhe	Johan	9876	15-04-1969	M	6214,19	7582	2
Selis	Noel	3456	20-08-1968	M	6214,19	7582	9
Dekocker	Veerle	6543	15-11-1974	V	5966,30	7582	4

Department

id_department	name	id_manager	manager_start
5	IWT	1234	01-sept-2013
1	VIVES	7582	01-sept-2012
9	OND	3456	01-jan-2015
2	HWB	9876	01-sept-2012
4	SAW	6543	01-sept-2014

Een kolom in een tabel herbergt gelijksoortige waarden welke **attribuutwaarden (attributen)** genoemd worden, de kop van een kolom bevat een attribuutnaam.

In een relationele databank moeten alle attribuutwaarden **atomair** zijn. Elke cel bevat één waarde. De waarden binnen een cel worden niet opgesplitst. Een cel kan geen lijst gegevens bevatten. Als een cel bijvoorbeeld [01-02-03] bevat, dan heeft dat één betekenis.

Attributen zijn ook **globaal** in het relationele model, d.w.z. als een attribuutnaam voorkomt in twee verschillende tabellen v/e relationele database, dan is de betekenis ervan in beide tabellen dezelfde. Vb. attribuutnaam `id_department` in de tabel `Employee` en in de tabel `Department`.

Bovendien moeten attributen **invariant zijn in de tijd**, de waarden moeten tijdsonafhankelijk zijn, e.g., sla niet de leeftijd van een persoon op in een databank, maar wel zijn geboortedatum. De waarde van het attribuut geboortedatum is tijdsonafhankelijk en de leeftijd kan altijd afgeleid worden.

Het aantal attributen in een tabel wordt **de graad van de tabel** genoemd, het aantal tupels de **cardinaliteit**. De cardinaliteit van een tabel kan in de loop van de tijd veranderen, de graad (meestal) niet. Samengevat heeft elke tabel in een relationele databank volgende eigenschappen:

- Elke kolom heeft een onderscheiden naam (attribuutnaam)
- De volgorde van de kolommen is niet van belang
- Een tabel bevat niet twee dezelfde rijen (een tabel is een verzameling van tupels en een verzameling kent per definitie geen duplicaten)
- De volgorde van de rijen is niet van belang (verzamelingen zijn immers niet geordend)
- Alle attribuutwaarden in de tabel zijn atomair

2.1.3 Domeinen

Met de term **domein** van een attribuut wordt de verzameling waarden bedoeld die een attribuut mag aannemen.

In de databank van Vives is het domein van het attribuut `sex` gelijk aan de verzameling `{M, V}`. Het domein van het attribuut `studgebnr` is een geheel getal van 1 tot 9 (grenzen incl.). Voor het attribuut `name` gaan we uit van een domein dat bestaat uit alle alfanumerieke waarden van maximaal 25 karakters. Dit betekent dat een naam een combinatie kan zijn van hoofdletters, kleine letters, leestekens, cijfers, enz. met een maximum van 25 tekens. Vb. `D'haese`.

Het domein van een attribuut is een **enkelvoudig domein** als alle elementen van het betreffende domein **atomair** zijn. In een relationele databank moet elk atribuut een enkelvoudig domein hebben.

Bijgevolg kan een attribuutwaarde nooit een samenstelling van waarden zijn, gedenormaliseerde data zijn immers niet toegestaan. In elke positie van een tabel van een relationele gegevensbank staat precies één waarde.

Het registreren van het feit dat een personeelslid van VIVES verbonden is aan twee studiegebieden van de hogeschool kan onmogelijk door in het attribuut `id_department` van de tabel Employee twee studiegebiednummers op te nemen.

2.1.4 Tupels

De gegevens in een rij van een tabel vormen samen het beeld van een object uit de werkelijkheid. Een rij is dus een verzameling bij elkaar horende attribuutwaarden, een dergelijke verzameling noemt men een **tupel** (of rij).

Een voorbeeld van een tupel uit de tabel `Department` is: `(2,HWB,9876,01-sept-2012)`.

Gezien de attribuutwaarden geen betekenis hebben zonder kennis van de bijpassende attributen vermelden we ook de verzameling attribuutnamen: `(id_department, name, id_manager, manager_start)`, dit wordt het **tupelschema** (relatieschema) genoemd.

2.1.5 Ontbrekende waarden (nulls)

Ontbrekende waarden zijn een belangrijk (en soms complex) concept in de gegevensbanktheorie en worden voorgesteld met een `null`. We geven een attribuut een `null`-waarde als:

- Een attribuutwaarde niet van toepassing is
- Een attribuutwaarde voor het moment nog onbekend is.

Een voorbeeld van de eerste situatie is de `null` waarde bij het attribuut `id_supervisor` van de algemeen directeur van VIVES: dhr. Hindryckx. Een illustratie van het tweede geval is het ontbreken van de geboortedatum van mevr. Beyls.

Belangrijk is dat een `null`-waarde niet verward wordt met het getal 0, dat is namelijk wel degelijk een waarde. Ook bij alfanumerieke attributen is een `null`-waarde formeel totaal verschillend t.o.v. leeg stuk tekst.

2.2 Sleutels

2.2.1 Kandidaatsleutels

De tupels van een tabel kunnen op een unieke manier van elkaar onderscheiden worden. Dit betekent dat ze op één of andere manier van elkaar moeten verschillen, via één attribuut of een groep attributen die voor elk tupel een andere waarde heeft. De attributenverzameling die voor elk tupel verschillend is, noemen we de **kandidaatsleutel** of **sleutel**.

De attributenverzameling bestaat uit één of meer attributen waarin geen `null`-waarden voorkomen, bovendien mag de attributenverzameling geen **overtollige attributen** bevatten.

Een kandidaatsleutel heeft de eigenschap van **minimaliteit**.

Voor de tabel `Employee` is het attribuut `name` geen kandidaatsleutel gezien meerdere personeelsleden dezelfde naam kunnen hebben. De combinatie `(name, id_employee)` is geen kandidaatsleutel omdat `name` een overtollig attribuut blijkt. De enige kandidaatsleutel die hier overblijft is `id_employee`.

In de veronderstelling dat in de tabel `Employee` ook het attribuut rijksregisternummer zou bestaan zijn er twee kandidaatsleutels: `id_employee` en `rijksregisternummer`.

Vermits elke tupel uniek is, bestaat er altijd minstens één kandidaatsleutel.

Als een sleutel uit één attribuut bestaat noemt men die sleutel een **enkelvoudige sleutel**. Een sleutel die uit een combinatie van attributen bestaat, heet een **samengestelde sleutel**.

2.2.2 Primaire sleutel (Primary Key)

De sleutel die door de gegevensbankontwerper daadwerkelijk gekozen wordt (uit de kandidaatsleutels) voor tupelidentificatie wordt de **primary key** genoemd.

Elke waardencombinatie van een primary key voor een tabel verwijst naar precies één tupel van die tabel. Gezien we een tupel in een tabel alleen kunnen herkennen als alle attribuutwaarden van de sleutel bekend zijn mag een waardencombinatie van een primary key in een tabel geen nulwaarden bevatten.

Een kandidaatsleutel die niet verkozen is tot primary key wordt een **alternatieve sleutel** (alternate key) genoemd.

Opdracht/Oefening: bepaal de primary key van de tabellen `Employee` en `Department`. Bepaal ook eventuele alternatieve sleutels.

2.2.3 Verwijssleutel (Foreign key)

Meestal houden de tabellen in een relationele gegevensbank verband met elkaar. Het komt bvb. voor dat een attributenverzameling voorkomt in twee relaties en dat de attributenverzameling de primary key is voor één van de relaties, maar niet voor beide.

Dan is er sprake van een **foreign key** (foreign key).

Het attribuut `id_department` is de primary key in de tabel `Department`, `id_department` komt echter ook voor in de tabel `Employee` en is geen primary key in deze tabel. In de tabel `Employee` vormt het attribuut `id_department` een foreign key, dit omdat het de sleutel is van een andere tabel en naar die tabel verwijst.

Verwijssleutels verwijzen van één tabel naar één of meerdere andere tabellen (soms naar dezelfde tabel). Op die manier zorgen ze voor samenhang tussen de verschillende tabellen van een gegevensbank.

De attribuutwaardencombinatie van een foreign key mag niet voor een gedeelte null-waarden bevatten (i.e. als de foreign key uit meerdere kolommen bestaat), de attribuutwaardencombinatie van een foreign key mag wel volledig uit null-waarden bestaan. Als dit laatste het geval is, verwijst de attribuutwaardencombinatie van de foreign key niet naar een andere tupel.

Zo zou in de tabel `Department` een studiegebied kunnen toegevoegd worden zonder aanduiding van de manager van dat studiegebied.

Opdracht/Oefening: bepaal alle foreign keys van de tabellen `Employee` en `Department`. Hou ook rekening met recursieve relaties.

2.2.4 Datastructuurdiagram/ERD

De tabellen en de relaties tussen de verschillende tabellen worden visueel weergegeven in een **datastructuurdiagram of ERD (entiteit-relatiediagram)**. Een ERD toont de entiteiten of objecten en hun relaties. Een voorbeeld van een entiteit is de entiteit `Klant`. `Klant` heeft dan een relatie met een andere entiteit, een `Bestelling`. Een klant heeft nul, een of meer bestellingen en een bestelling werd geplaatst door één klant. Op technisch databaseniveau spreken we dan over verbanden tussen tabellen via de foreign key.

Er zijn diverse mogelijke voorstellingswijzen van een ERD. Naast de bekende Chen-notatie is er ook de kraaiepootnotaties (**Crow's foot notation**), die we in deze cursus gebruiken. Deze naam heeft te maken met de notatiewijze van relaties zoals hieronder zal blijken.

We kunnen een ERD ontwerpen op diverse niveaus:

- Op het *conceptuele* niveau ontwerpen we de gegevensstructuur nog op een vrij abstract niveau waarbij we de informatiebehoeften van onze klant en de relaties tussen entiteiten in kaart brengen zonder rekening te houden met de beperkingen van een relationele database. Op dat niveau zullen we bijvoorbeeld veel-op-veel-relaties toelaten en zullen we de foreign keys en primary keys nog niet definiëren.

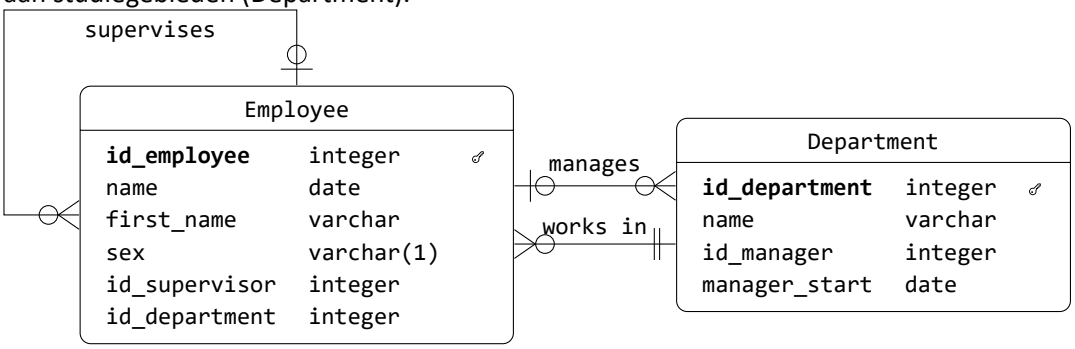
- Op het *logische* niveau zullen we wel elke entiteit/tabel en elke relatie in kaart brengen en de attributen duidelijk definiëren met hun respectievelijke logische datatypes (tekst, geheel getal, kommagetal, booleaanse waarde, media-object) zonder er specifiek rekening mee te houden in welke technologie we onze relationele database zullen definiëren. We zullen dus wel de primary keys, alternatieve sleutels, foreign keys aanduiden, maar we zullen er ons nog niet om bekommeren of het systeem een kommagetal definieert als numeric, float of een ander datatype specifiek voor deze technologie.

- Op het *fysieke* niveau houden we rekening met de RDBMS die we zullen hanteren. Het model bevat alle details opdat de database automatisch gegenereerd kan worden.

Wij zullen in deze cursus ERD's op het logische niveau tekenen.

De relatie tussen twee entiteiten wordt soms aangeduid met de term **'ouder/kind-associatie'**. De tabel met de foreign key heet dan de kindtabel, de tabel waarnaar verwezen wordt heet de oudertabel.

Datastructuurdiagram van de VIVES-databank dat personeelsleden (Employee) koppelt aan studiegebieden (Department).




```
Employee(id_employee, surname, name, birthdate, sex, pay, id_supervisor,
    id_department)
Department(id_department, name, id_manager, manager_start)
```

```
FK van Employee(id_supervisor) naar Employee(id_employee)
FK van Department(id_manager) naar Employee(id_employee)
FK van Employee(id_department) naar Department(id_department)
```

Het datastructuurdiagram geeft, samen met een opsomming van de attributen per tabel, een duidelijk beeld van de structuur van een database. De naam van de relatie moet kort en bondig zijn en duidelijk de relatie beschrijven.

- Een personeelslid 'works in' een studiegebied:
  `FK van Employee(id_department)naar Department(id_department).`
- Een personeelslid 'manages' een studiegebied:
  `FK van Department(id_manager)naar Employee(id_department).`
- Een personeelslid 'supervises' een (ander) personeelslid:
  `FK van Employee(id_supervisor)naar Employee(id_employee).`

Doordat de tabelstructuur al dan niet toelaat dat je meerdere connecties maakt via zulke relaties worden de zogenaamde cardinaliteiten gebruikt om dit duidelijk te maken (zie Figuur 2.2). Cardinaliteiten moet men in het schema steeds aflezen 'aan de andere kant' van de verbindingslijn, lezend van de ene tabel naar de andere.

Voor wat de relatie 'works in' betreft, vormt `Department` de oudertabel en `Employee` de kindtabel (bevat de foreign key). Er wordt expliciet aangegeven dat er:

- Bij één rij van `Department`, nul of meer rijen van `Employee` horen.
- Bij één rij van `Employee`, precies één rij van `Department` hoort.

Dit betekent dat een personeelslid steeds moet verbonden zijn aan een studiegebied, m.a.w. de het attribuut `(id_department)` (er vertrekt hier een foreign key naar `Department(id_department)`) in `Employee` *moet* verplicht ingevuld worden! De realisatie hiervan gebeurt met een extra beperkende voorwaarde op de foreign key (not-null constraint, zie later).

Bij de relatie 'manages' vormt `Employee` de oudertabel en `Department` de kindtabel (bevat de foreign key).

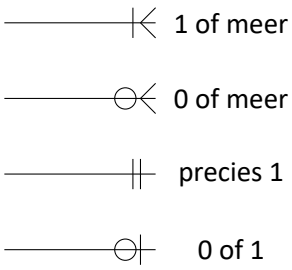
- Bij één rij van `Employee` (de eventuele manager) horen nul of meer `Department`-rijen.
- Bij één rij van `Department` hoort nul of één `Employee`-rij (de manager).

Dit betekent dat een studiegebied kan toegevoegd worden zonder aanduiding van de manager, m.a.w. het attribuut `id_manager` mag `null` zijn (terwijl er een foreign key uit vertrekt). Gezien, volgens de relationele theorie, de attributen waaruit een foreign key vertrekt (allemaal) `null` mogen zijn (zie Sectie 2.2.3) impliceert dit geen extra voorwaarde op deze foreign key. Men spreekt in dit verband ook van een optionele (niet-verplichte) relatie.

De relatie 'supervises' is in die zin speciaal dat de verwijzing naar de tabel zelf gebeurt, men noemt dit een recursieve relatie.

- Bij één rij van `Employee` hoort geen of 1 rij van `Employee` die de chef aanduidt
- Bij één rij van `Employee` horen 0 of meer rijen van `Employee` die de ondergeschikten aanduiden.

De verschillende cardinaliteitsnotaties in een datastructuurdiagram kan je in Figuur 2.2 terugvinden. Merk op dat sommige cardinaliteitsratios moeilijker af te dwingen zijn in een relationele databank, zoals de 'één-of-meer' relatie. Typisch kan je dit bereiken met extra constraints, gecombineerd met het tegelijkertijd aanpassen van meerdere tabellen in één transactie. Zulke situaties zijn vaak specifiek en zullen afhangen van de requirements.



Figuur 2.2: Notaties van de verschillende cardinaliteitsratios.

Opgelet! Programma's zoals Visual Paradigm volgen de regels van de ERD notatie niet tot op de letter. Visual Paradigm gebruikt voor 'precies 1' één streepje i.p.v. twee, en voor '0 of 1' een 0 i.p.v. een 0 en een streepje.
In deze cursus zal je soms misschien ook de enkelvoudige notatie zien, wees je dan bewust van de verschillen tussen deze notaties en wat ze betekenen!

2.3 Integriteit

De gegevens van een gegevensbank moeten steeds **consistent** (met elkaar in overeenstemming) zijn. Het mag bvb. niet zo zijn dat een gegevensbank twee tupels bevat, waarvan de ene aangeeft dat een persoon gedomicilieerd is in Kortrijk en de andere dat dezelfde persoon gedomicilieerd is in Knokke.

De grootste bedreiging voor de consistentie van een informatiesysteem vormt de **redundantie** (overtolligheid). Er moet ten allen tijde naar gestreefd worden dat een bepaald gegeven (vb. adres van een persoon) maar éénmalig gestockeerd wordt. Om dat te bereiken zullen we ernaar streven een genormaliseerde database te ontwerpen. Dat wordt in het volgende hoofdstuk behandeld.

Ook het opslaan van zogenaamde afleidbare gegevens is een vorm van redundantie en dient vermeden te worden. Indien een bedrag (excl. BTW) en het BTW-percentage opgeslagen worden is het overbodig het bedrag inclusief BTW op te slaan. Soms wordt echter selectief besloten tot het opslaan van redundante gegevens; vooral in gevallen waar snelheid (performance) cruciaal is, en waarbij het steeds opnieuw berekenen of afleiden van de gewenste gegevens teveel tijd zou kosten.

Bovendien moeten de gegevens correct zijn. We willen dus dat de gegevens in een gegevensbank volledig en correct zijn, ofwel: dat de **integriteit** van een gegevensbank gegarandeerd is.

Teneinde de integriteit van een gegevensbank zo goed als mogelijk te garanderen worden **integriteitsregels** (constraints) opgesteld.

2.3.1 Entiteit-integriteit (Entity integrity)

In een primary key van een tabel mogen geen attributen met nulwaarden voorkomen.

Elke tupel moet uniek worden geïdentificeerd door een waardencombinatie van de primary key. Het toestaan van een `null`-waarde in de primary key zou bijgevolg een contradictie betekenen gezien een `null`-waarde de afwezigheid van een waarde voorstelt.

Per definitie is een primary key ook uniek.

2.3.2 Referentiële integriteit (Referential integrity)

In een relationele database mag een waardencombinatie van een foreign key niet verwijzen naar een tupel die niet bestaat. Als een tupel t_2 verwijst naar een tupel t_1, dan *moet* t_1 bestaan. Als het tupel t_2 daarentegen niet verwijst naar een andere tupel, dan moeten alle attribuutwaarden van de foreign key `null`-waarden zijn. Deze referentiële restrictie staat bekend als de referentiële integriteitseis.

2.3.3 Domeinrestricties (Domain constraints)

Een **domeinrestrictie** geeft aan dat een attribuut is gedefinieerd op een specifiek domein.

Het definiëren dat de waarde van het attribuut `pay` groter dan 0 moet zijn, of dat de waarde van het attribuut `sex` `M` of `V` moet zijn, of dat de naam geen `null`-waarde mag zijn, zijn voorbeelden van domeinrestricties.

Algemeen noemen we beperkende voorwaarden met betrekking tot de toegestane inhoud van een database constraints. Door de juiste constraints te definiëren kunnen we de eisen met betrekking tot de integriteit vastleggen.

Met behulp van SQL kan men volgende constraints definiëren:

- `primary key` constraints
- `foreign key` constraints
- `unique` constraints (voor alternate keys)
- `not null` constraints (voor verplichte velden)
- `check` constraints (domeinrestricties, afhankelijkheden bvb. `birthdate < contract_start`)

2.4 Relationele algebra

Relationele algebra is een formele taal waarmee we onze onze databank kunnen bevragen. We zullen m.a.w. in algemene regels definiëren hoe we een bepaald resultaat willen bereiken, rekening houdend met de structuur van de database. Wanneer we bijvoorbeeld van de klanten uit Kotrijk de lijst van orders willen krijgen, kunnen we via de relationele algebra definiëren hoe we tot dat resultaat kunnen komen. We doen dit in een formele taal die rekening houdt met de structuur van relationele databanken, maar die voor de rest onafhankelijk is van een concrete querytaal. In concrete systemen hebben we querytalen zoals SQL, Xquery, Linq query, enzovoort die deze relationele algebra concreet in praktijk brengen.

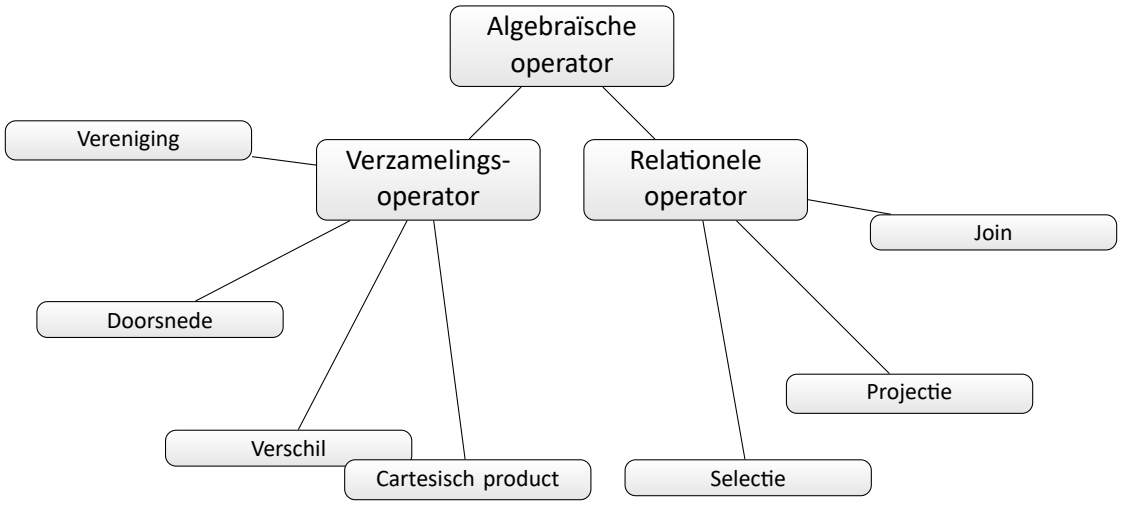
Bij het rekenen maken we gebruik van zogeheten rekenkundige operatoren. Bekende voorbeelden daarvan zijn $+$, $-$, $*$, $/$, $\sqrt{\ }$.

Relationele algebra is een eenvoudige formele taal die een aantal operatoren bevat die we **algebraïsche operatoren** noemen. Een algebraïsche operator kan één operand hebben (unaire operator) of twee operanden (binaire operator). Daarbij is een operand altijd een tabel.

Een algebraïsche operator is een **verzamelingsoperator** of een **relationele operator** (zie Figuur 2.3).

De verzamelingenoperatoren zijn bekend vanuit de verzamelingenleer. Bekende verzamelingenoperaties zijn: **vereniging, doorsnede, verschil** en productverzameling of **cartesisch product**. Gezien het feit dat een tabel bestaat uit een verzameling tupels kunnen verzamelingenoperatoren toegepast worden op tabellen.

Relationele operatoren werken niet op willekeurige verzamelingen, wel op tabellen. Volgende relationele operaties komen aan de orde: **selectie, projectie, join**.



Figuur 2.3: Algebraïsche operatoren

In de praktijk zullen deze bewerkingen uitgevoerd worden met de relationele taal SQL (Structured Query Language).

2.4.1 Selectie

De **selectie-operator** is een relationele operator die tot doel heeft het selecteren van tupels uit een tabel. Bij een **selectie** hoort een selectievoorwaarde. De selectievoorwaarde kan een **enkelvoudige formule** zijn: hierbij wordt een attribuutwaarde vergeleken met een constante, of worden twee attribuutwaarden uit dezelfde tabel met elkaar vergeleken. Daarbij maken we gebruik van de **vergelijkingsoperatoren**: $=, <>, <, <=, >, >=$.

```
pay > 5000
sex = 'V'
```

Een aantal enkelvoudige voorwaarden kunnen we samenvoegen tot een meervoudige. We koppelen twee voorwaarden aan elkaar met behulp van de logische operator and (conjunctie-operator) of de logische operator or (disjunctie-operator).

```
pay >= 5000 and pay <= 6250
postnr >= 8000 and postnr < 9000
```

De and-operator heeft prioriteit op de or-operator. De volgende twee notaties zijn equivalent:

```
postnr >= 3000 and postnr < 4000  or postnr > 9000
(postnr >= 3000 and postnr < 4000) or postnr > 9000
```

We kunnen ook gebruik maken van de logische operator not (ontkenningsoperator). Als we deze plaatsen voor een voorwaarde, dan wordt de oorspronkelijke voorwaarde ontkend. Let erop dat de ontkenningsoperator een hogere prioriteit heeft dan de logische and (en uiteraard dan de logische or).

De volgende twee notaties zijn niet equivalent:

```
not  pay >= 5000 and pay <= 6250
not (pay >= 5000 and pay <= 6250)
```

Het toepassen van de selectie-operator resulteert in een verzameling tupels met hetzelfde tupelschema. Een dergelijke verzameling tupels noemen we een selectie. Dus het resultaat van de operatie noemt hetzelfde als de operatie zelf.

Een selectiebewerking met de selectievoorwaarde uit het laatste voorbeeld geeft volgende selectie:

name	id_employee	birthdate	sex	pay	id_supervisor	id_department
Ketels	8167	12-04-88	M	4602,88	7365	2
Haegeman	1234	31-12-70	M	6718,40	7582	5
Hindryckx	7582	01-01-60	M	8197,34	null	1
Beyls	6741	null	V	4478,94	7365	2

2.4.2 Projectie

Een selectieoperator licht een aantal rijen (tupels) uit een tabel. De **projectie-operator** daarentegen licht een aantal kolommen (attributen) uit een tabel, en voegt ze samen tot een nieuwe tabel, waarna de identieke rijen worden verwijderd (een tabel mag geen identieke rijen bevatten). Het aantal rijen van de projectie is bijgevolg steeds kleiner of gelijk aan het aantal rijen van de originele tabel.

Aan een projectie-operator is een verzameling gekoppeld, bestaande uit attribuutnamen waarmee de kolommen worden aangegeven die uit de tabel moeten worden gelicht.

Een projectiebewerking op de tabel `Employee` m.b.v. de verzameling attributen `{id_manager, id_department}` geeft volgend resultaat:

id_manager	id_department
7365	2
9876	2
7582	5
Null	1
7582	2
7582	9
7582	4

Merk op dat de waarden `{3765,2}` slechts één keer voorkomt in de lijst. Gelijke waarden komen maar één keer voor.

2.4.3 Join

Bij relationele gegevensbanken neemt de **join-operator** een heel belangrijke plaats is. Met behulp van de join-operatie is het mogelijk om gegevens uit verschillende tabellen te combineren aan de hand van gemeenschappelijke attributen en attribuutwaarden.

De meest voor de hand liggende **join (natuurlijke join, equi-join)** is deze waarbij uit twee (of meer) tabellen één resultaattabel wordt afgeleid op basis van een **vergelijkingsattribuut** (meestal de foreign key): alle mogelijke combinaties van een rij uit de ene tabel met een rij uit de andere tabel worden gemaakt, mits beide rijen voor het vergelijkingsattribuut dezelfde waarde hebben. Een combinatie van twee rijen ontstaat hierbij door ze aan elkaar te plakken, en de dubbele attribuutwaarde slechts éénmaal in de resultaatrij op te nemen.

Een join tussen `Employee` en `Department` op basis van het attribuut `id_department` kan je hieronder zien. Niet alle kolommen worden getoond.

name	id_employee	id_supervisor	id_department	department_name	id_manager
Johan	6541	7365	2	HWB	9876
Marc	4379	7365	2	HWB	9876
Bavo	8167	7365	2	HWB	9876
Arne	7365	9876	2	HWB	9876
Wim	1234	7582	5	IWT	1234
Joris	7582	Null	1	VIVES	7582
Katrien	6741	7365	2	HWB	9876
Johan	9876	7582	2	HWB	9876
Noel	3456	7582	9	OND	3456
Veerle	6543	7582	4	SAW	6543

Merken we nog op dat het combineren van elke rij uit de ene tabel met elke rij uit de andere tabel zonder een vergelijkingsattribuut te gebruiken neerkomt op het **cartesisch product** tussen twee tabellen. Het cartesisch product tussen `Employee` en `Department` zou resulteren in een tabel van 50 rijen (10 x 5) en zinloze informatie opleveren.

Wel kan gesteld worden dat een (natuurlijke) join neerkomt op het achtereenvolgens toepassen van de operatoren: cartesisch product, selectie en projectie.

In bepaalde gevallen is het ook zinvol een tabel te joinen met zichzelf. Er is een aparte naam voor, hoewel er geen essentieel verschil is met een gewone join. We noemen zo'n join een **auto-join** of **self-join**. De gegevens uit een tabel worden gecombineerd met de gegevens van een (fictieve) kopie van dezelfde tabel, dit door vergelijking van een attribuut uit de tabel met een attribuut uit de (fictieve) kopie van de tabel.

Hieronder een interessant voorbeeld van een auto-join op de tabel `Employee`. Dit produceert een overzicht van de medewerkers met daarbij de naam van hun chef.

surname	id_employee	manager
Acx	6541	Vandenbussche
Desplenter	4379	Vandenbussche
Ketels	8167	Vandenbussche
Vandenbussche	7365	De Langhe
Haegeman	1234	Hindryckx
Beyls	6741	Vandenbussche
De Langhe	9876	Hindryckx
Selis	3456	Hindryckx
Dekocker	6543	Hindryckx

Het attribuut `id_manager` uit de ene tabel wordt vergeleken met het attribuut `id_employee` uit de (fictieve) kopie van die tabel. Bemerk dat van de 10 medewerkers er slechts 9 in het overzicht voorkomen.

De `null`-waarde van het attribuut `id_manager` voor de algemeen directeur Hindryckx betekent dat deze waarde niet van toepassing is, deze waarde kan dan ook niet gecombineerd worden met een Persnr. Als we het feit dat algemeen directeur Hindryckx wel degelijk een medewerker van VIVES is, maar geen manager binnen VIVES meer heeft, expliciet tot uiting willen brengen, kunnen we dit doen met een zogeheten **outerjoin**.

Het resultaat van de outerjoin ziet er dan als volgt uit:

surname	id_employee	manager
Acx	6541	Vandenbussche
Desplenter	4379	Vandenbussche
Ketels	8167	Vandenbussche
Vandenbussche	7365	De Langhe
Haegeman	1234	Hindryckx
Hindryckx	7582	
Beyls	6741	Vandenbussche
De Langhe	9876	Hindryckx
Selis	3456	Hindryckx
Dekocker	6543	Hindryckx

Merk de lege cell op bij Hindryckx, waar een null-waarde wordt getoond.

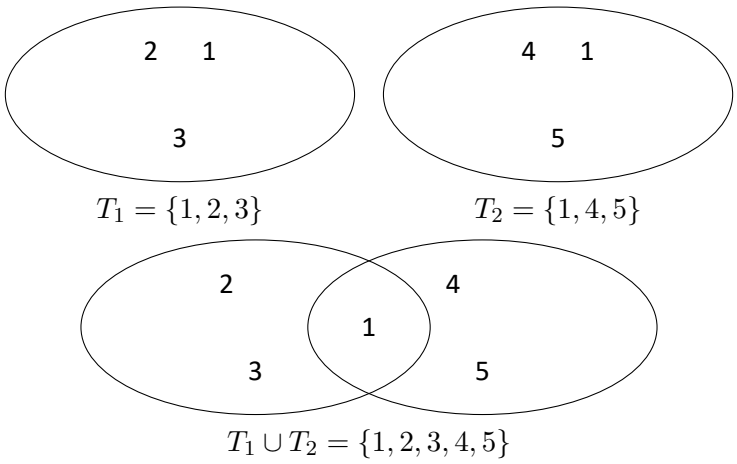
Bij een **outerjoin** wordt een rij uit een tabel die bij het joinen niet kan gecombineerd worden met een rij uit de andere tabel op basis van dezelfde attribuutwaarden toch in het eindresultaat opgenomen.

2.4.4 Vereniging (Union)

De vereniging of unie van twee tabellen is op dezelfde manier gedefinieerd als gebruikelijk is in de verzamelingenleer. Voorwaarde is dat de tabellen **vergelijkbaar** zijn, dit betekent dat de tabellen

evenveel attributen tellen (zelfde graad) en dat de overeenkomstige attributen hetzelfde domein hebben.

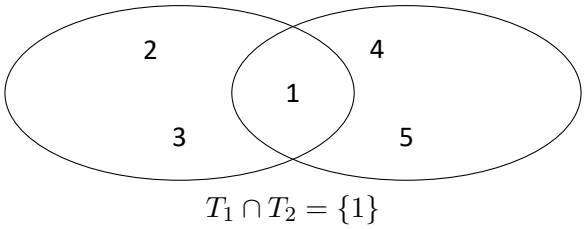
Het resultaat van de vereniging van 2 vergelijkbare tabellen is de verzameling van alle tupels die in tabel1 of in tabel2 of in beide tabellen voorkomen. Het resultaat bevat geen duplicaatrijen. De unie operator is geïllustreerd in Figuur 2.4.



Figuur 2.4: Illustratie unie operator.

2.4.5 Doorsnede (Intersect)

Je mag de doorsnede operator enkel gebruiken wanneer de tabellen vergelijkbaar zijn. De doorsnede van 2 vergelijkbare tabellen is de verzameling van alle tupels die zowel in tabel1 als in tabel2 voorkomen. Het resultaat bevat geen duplicaatrijen. Figuur 2.5 illustreert het principe.



Figuur 2.5: Illustratie intersect operator.

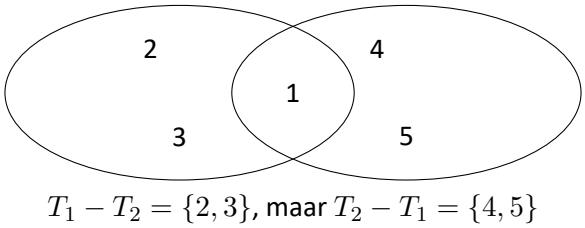
2.4.6 Verschil (minus)

De functie van de **verschiloperator** is het selecteren van de tupels van een tabel welke wel in de tabel, maar niet in een andere tabel voorkomen. Voorwaarde is eveneens dat de tabellen vergelijkbaar zijn.

In tegenstelling tot unie en intersectie is het verschil asymmetrisch: $(T_1 – T_2) \neq (T_2 – T_1)$.

2.4.7 Views

Bij een combinatie van projecties, selecties, joins, …verkrijgen we een resultaat dat ook een tabel is. Deze “kijk” op de gegevens kunnen we ook permanent maken en als een virtuele tabel opslaan.



Figuur 2.6: Illustratie minus operator.

Dat noemen we een view.

De bekomen informatie wordt echter niet fysiek in de database opgeslagen. Views zijn te beschouwen als virtuele tabellen, een wijziging van gegevens in een tabel wijzigt dus ook gegevens in een view die ervan afgeleid is.

Men kan views net als tabellen benaderen en manipuleren met behulp van SQL-commando's. Vaak weet een gebruiker niet of hij een tabel, dan wel een view op een tabel benadert.

Onderstaande view ontstaat door een projectie-operatie op de tabel Employee toe te passen waarbij de kolommen `pay` en `name` geweerd wordt. Het produceert een `VIEW` op `Employee`.

surname	id_employee	birthdate	sex	id_manager	id_department
Acx	6541	15-jan-63	M	7365	2
Desplenter	4379	19-feb-62	M	7365	2
Ketels	8167	12-apr-88	M	7365	2
Vandenbussche	7365	29-feb-68	M	9876	2
Haegeman	1234	31-dec-70	M	7582	5
Hindryckx	7582	1-jan-60	M	null	1
Beyls	6741	null	V	7365	2
De Langhe	9876	15-apr-69	M	7582	2
Selis	3456	20-aug-68	M	7582	9
Dekocker	6543	15-nov-74	V	7582	4

Het gebruik van views is vooral in verband met beveiliging en controle belangrijk, gevoelige gegevens kunnen op die manier afgeschermd worden tegen onbevoegd gebruik.

In bepaalde gevallen kan een complexe query tevens eenvoudiger opgelost worden door tussenresultaten als views te definiëren.

Laat ons dit illustreren met een concreet **voorbeeld.** Uit onze voorbeelddatabase van VIVES willen we de naam en voornaam van alle mannelijke personeelsleden die verbonden zijn aan het departement HWB.

We kunnen dit in diverse stappen realiseren.

- Om te beginnen hebben we gegevens uit twee tabellen nodig. De naam en de voornaam van een werknemer zit in de tabel `Employee` en de naam van het studiegebied in `Department`. We zullen dus een join nodig hebben. Het resultaat van deze operatie zullen we de view `Employee_Department` noemen.

- Op het resultaat van die join zullen we selectie toepassen, namelijk die personeelsleden waarvan de naam van het studiegebied gelijk is aan `'HWB'` en het geslacht gelijk is een `'M'`. De view die het resultaat is van die operatie zullen we `Employee_Department_HWB` noemen.
- Op dit resultaat zullen we een projectie toepassen, om enkel de kolommen naam en voornaam over te houden. Dit resultaat zullen we `Result` noemen.

Dit geeft dus:

- `Employee_Department = Employee join Department` met joinvoorwaarden `Employee.id_department = Department.id_department`.
- `Employee_Department_HWB` = selectie op `Employee_Department` met selectievoorwaarde: `(Department.name = 'HWB' AND Employee.sex = 'M')`.
- `Result` = projectie op `Employee_Department_HWB` met kolommen `(Employee.surname, Employee.name)`.

Als bovenstaande vraag geregeld gesteld worden, dan maken we die permanent en definiëren we die als een view. Je kan nu een eenvoudige opvraging doen op de virtuele tabel `Result`, op de view `Result`. Je krijgt als resultaat een eenvoudige lijst gegevens waarachter in feite een complexe reeks operaties zit.

2.5 Oefeningen

2.5.1 Begrippen

Omschrijf met eigen woorden de volgende begrippen:

- Consistentie
- Redundantie
- Integriteit
- Attribuut/attribuutwaarde
- Domein
- Tupel/tupelschema
- Tabel
- Query
- View
- Relationele databank
- Primaire sleutel
- Verwijssleutel

2.5.2 IQ – lengte - leeftijd

Het CLB (centrum voor leerlingenbegeleiding) doet een onderzoek naar het verband tussen intelligentiequotiënt (IQ) en de lengte en leeftijd bij kinderen van de basisschool. Een tabel uit de databank kan je in Tabel 2.1 zien.

- Zijn alle attributen atomair?
- Definieer geschikte domeinen voor de verschillende attributen.
- Is het attribuut leeftijd invariant in de tijd?
- Welk probleem kan er ontstaan en hoe is het op te lossen?

height	age	iq
112	6	90
130	8	103
115	7	98
145	12	124
126	7	95
127	8	103
118	6	114
141	10	90
118	6	110
142	11	117
104	6	105

Tabel 2.1: Tabel uit CLB databank.

2.5.3 Studenten en hun vakken

Gegeven een databank met drie tabellen over studenten en hun vakken (zie Tabel 2.2).

- Bepaal de primaire-, de alternatieve- en de foreign keys?
- Teken het datastructuurdiagram.
- Voer een join-operatie uit met respectievelijk: `Student` en `Subscription`; `Course` en `Subscription`; `Student` en `Subscription` en `Course`.
- Bepaal hoe volgende informatie in de database geregistreerd wordt: “De nieuw ingeschreven student `'Devos'` zal het vak `'Oracle'` volgen”.
- Voeg de volgende tupel toe aan de tabel `Course`: `(40,'C++')`.
- Voeg de volgende tupels toe aan `Subscription`: `(70,70)` en `(70,null)`.
- Hoe wordt student ‘Verschuere’ uitgeschreven?

id_student	name_student
10	Declercq
30	Naessens
20	Vandaele
40	Decroos
70	Verschuere
50	Dewaele

Tabel Student

id_course	name_course
50	Concepten databanken
10	Oracle
40	Dba
30	Sql

Tabel Course

id_student	id_course
30	40
30	10
30	30
70	50
30	50
70	30
50	50

Tabel Subscription

Tabel 2.2: Studenten en hun vakken.

2.5.4 Bestellingen in de bistro

Een bistro neemt zijn bestelling op via een tabel. Ze noteren het tafelnummer, de bestelde producten en de bestelde hoeveelheid van elk product. De software registreert automatisch het tijdstip van de bestelling en berekent de totale prijs van de bestelling, gebaseerd op een lijst van producten en hun prijs. Als de rekening betaald werd, wordt dit ook in de computer geregistreerd. Elke bestelling heeft een uniek nummer en elk product een unieke productcode. De database ziet er als volgt uit:

```
Order (id_order, timestamp, table_number, total_price, paid)
Orderline (id_order, id_product, quantity)
Product (id_product, name, price)
```

Maak de volgende oefeningen:

- Onderlijn de primary keys.
- Schrijf FK onder de foreign keys.
- Teken het ERD.

2.5.5 Handbalcompetitie

We willen gegevens opslaan van een handbalcompetitiereeks van één seizoen. Elk team in die reeks heeft een uniek nummer en een unieke naam. We slaan ook de naam en het telefoonnummer op van de coach en van de club van dat team.

Elk team speelt een aantal wedstrijden tegen de andere teams van de reeks. Elke match heeft een unieke code. Voor elke match willen we weten welk van de twee teams de wedstrijd speelden, de datum en het tijdstip van elke match, de thuisploeg, de bezoekende ploeg, de precieze locatie en het resultaat.

We houden ook gegevens bij over de spelers van elk team. Elke speler behoort tot één team. We hebben het officiële lidnummer van elke speler. Deze code is uniek voor elke speler in de

competitie. De naam, het adres en de geboortedatum van elke speler is bekend. Voor elke match kennen we de deelnemende spelers, het aantal doelpunten en zijn shirtnummer.

De database heeft de volgende structuur:

```
Team (id_team, name, coach, coach_phone, id_club)
Match (id_match, id_home_team, id_away_team, timestamp, location, result)
Player (id_player, name, street, postal_number, city, birthdate, id_team)
Participation (id_player, id_match, goals, shirt_number)
```

Maak de volgende oefeningen:

- Onderstreep de primary keys.
- Schrijf FK onder elke foreign key.
- Teken het ERD.
- Wat moet er gebeuren om een team volledig te wissen?

2.5.6 Relationele algebra en de VIVES-database

Voer de volgende operaties uit op de VIVES-databank. Gebruik relationele algebra. Werk stap voor stap met tussentijdse views.

1. Schrijf voor de VIVES-databank de selectievoorwaarde en het resultaat van de selectie op de tabel `Employee` waarbij alle personeelsleden worden geselecteerd die tussen €5000 en €6000 verdienen (grenzen inbegrepen).
2. Wat is het resultaat van een projectie `Supervisor` op de tabel `Employee` a.d.h.v. de attribuutverzameling `(id_supervisor)`?
3. Maak een projectie `Id_name` op de tabel `Employee` a.d.h.v. de attribuutverzameling `(id_employee,surname)`.
4. Voer een join-operatie uit m.b.v. bovenstaande projecties `Supervisor` en `Id_name`.
5. Voer een projectie `Emp` uit op de tabel `Employee` a.d.h.v. de attribuutverzameling `(id_employee)`.
6. Bepaal met de projecties `Emp` en `Supervisor` en een passende verzamelingenoperator het personeel dat geen leidinggevende functie heeft.
7. Waaraan is de unie van `Emp` en `Supervisor` gelijk? Verklaar?
8. Waaraan is het verschil `Supervisor` – `Emp` gelijk? Verklaar?
9. Beschrijf stap voor stap hoe ik met relationele algebra de namen en voornamen van de vrouwelijke personeelsleden kan ophalen.
10. Beschrijf stap voor stap hoe ik de naam kan ophalen van de manager van studiegebied IWT.
11. Beschrijf stap voor stap hoe de namen van de personeelsleden kan ophalen die Joris Hindrycks als chef hebben.

2.5.7 Studiecontracten

Aan een hogeschool worden met de studenten studiecontracten afgesloten. Een studiecontract van een student geeft aan welke vakken een student in het academiejaar opneemt (meestal vakken voor een totaal van 60 studiepunten).

Om de informatie bij te houden omtrent de studiecontracten van een bepaald academiejaar maken we gebruik van volgende tabellen:

```
Student (id_student, email, surname, name, class_id)
Class (id_class, name)
Study_program (id_student, remark, number_absences, end_result, id_course)
Course (id_course, name, study_points)
```

Gevraagd:

1. Geef voor elke tabel de **primary key**?
2. Geef voor elke tabel de **foreign key(s)**?
3. Teken het **datastructuurdiagram** inclusief **cardinaliteiten** en **namen van relaties**.
4. Bespreek stap voor stap hoe je via relationele algebra de namen en voornamen kunt bepalen van studenten uit de klas met code 3APP .
5. Bespreek stap voor stap hoe ik van alle studenten de volgende informatie wil ophalen: `(id_student, name, surname, class_name)`.
6. Bespreek stap voor stap hoe de informatie van een studiecontract van een bepaalde student, op basis van het studentnr, uit de database kan opgehaald worden. Maak hierbij gebruik van verzamelings- en/of relationele operatoren.
   Voorbeeld studiecontract student 587468:

Databaseconcepten	3
Concepten OO 1	5
Lab Java 1	4

7. Bespreek stap voor stap hoe de studentnummers kunnen bepaald worden van de studenten welke nog geen studiecontract (m.a.w. geen enkel vak opgenomen hebben) afgesloten hebben. Maak hierbij gebruik van verzamelings- en/of relationele operatoren.
8. Bespreek nauwgezet hoe u tewerk gaat om alle informatie van één bepaalde student uit de database te verwijderen.

2.5.8 CAR-PASS

Om de fraude met kilometerstanden bij tweedehandswagens te bestrijden werd sedert 1 december 2006 het car-pass ingevoerd.

De opzet is als volgt: bij onderhoud van een wagen in een garage, bij een bandenwissel in een bandencentrale, bij technische keuring, enz… moet de kilometerstand van de betreffende wagen op de betreffende datum doorgegeven worden aan de vzw CAR-PASS en wordt er in een database opgeslagen. Wanneer de wagen verkocht wordt, is de verkoper (particulier of handelaar) verplicht een car-pass ter beschikking te stellen, dit wordt via de technische keuring aangevraagd. Het car-pass bevat de kilometerhistoriek van het voertuig, op die manier wordt gesjoemel met kilometertellers vrijwel uitgesloten:

Een kilometerstand op een bepaalde datum welke lager of gelijk is aan de kilometerstand op een vorige datum zal op het car-pass direct argwaan opwekken!

Opmerkingen bij de tabel Registration:

- Mogelijke codes zijn: 'O': onderhoud, 'K': keuring, 'B': bandencentrale.
- Wanneer rijksregisternummer eigenaar niet ingevuld is betreft het een voertuig van een garage.
- Uiteraard kan een voertuig (meermaals) veranderen van eigenaar / nummerplaat.

Veronderstel dat na normalisatie tot de derde normaalvorm volgende tabellen ontstaan:

```
Car (brand, model, chassisnumber, cylinder_volume, date_registration,
    date_first_use)
Registration (kilometers, timestamp, id_registration, plate, chassisnumber,
    citizen_id_owner)
Owner (citizen_id, name, address, postal_code, town)
```

- Geef voor elke tabel de **primary key**?
- Geef voor elke tabel de **foreign key(s)**?
- Teken het **datastructuurdiagram** inclusief **cardinaliteiten** en **namen van relaties**.
- Bespreek stap voor stap hoe ik de chassisnummers van de auto's kan ophalen die voor 1 januari 2019 ingeschreven zijn.
- Bespreek stap voor stap hoe ik de datums kan ophalen wanneer de auto met nummerplaat `2-EXD-643` onderhoud heeft gehad.
- Bespreek stap voor stap hoe ik de namen kan ophalen van eigenaars van de auto met chassisnummer: `1HGBH41JXMN109186`.
- Bespreek stap voor stap hoe de chassisnummers kunnen bepaald worden van de auto's welke nog geen kilometerregistratie gehad hebben tijdens een keuring. Maak hierbij gebruik van verzamelings- en/of relationele operatoren.
- Bespreek nauwgezet hoe u tewerk gaat om alle informatie van één bepaalde auto uit de database te verwijderen.

2.5.9 Hotelreservaties

In een toeristische dienst wil men gegevens bijhouden omtrent het reserveren van kamers.

Van ieder hotel wordt een uniek identificatienummer (id_hotel), de naam van het hotel, de straat en nummer, id van de stad en het aantal sterren bijgehouden. Ieder hotel beschikt over een aantal kamers. Iedere kamer krijgt een kamernummer dat uniek is per hotel. Van een kamer wordt naast het kamernummer ook nog het hotelid, de prijs per nacht, het maximum aantal personen per kamer, het aantal nachten dat deze kamer reeds verhuurd is dit jaar en de datum laatst verhuurd bijgehouden.

Een klant kan nu één of meerdere kamers boeken in één of meerdere hotels. Iedere klant heeft een uniek klantid. Verder wordt ook zijn naam, voornaam, geboortedatum en geslacht bijgehouden.

Iedere boeking krijgt een uniek boekingid. Per boeking wordt nog het kamernummer, het hotelid, het klantid, de datum van aankomst, het aantal nachten en is de boeking al of niet als betaald genoteerd.

Er zijn nog 2 tabellen met informatie over de steden en de regio's (streken) waarin de steden liggen. De tabel regio bevat een uniek regioid, de naam van de regio en het id van de hoofdstad

van de regio. De tabel stad bevat een uniek stadid, de naam van de stad en het regioid waarin de stad gelegen is.

Een bepaalde kamer kan tijdelijk onbeschikbaar zijn in een hotel. In dat geval wordt er een boekingrecord gemaakt, waarbij de velden id_klantid en betaald niet ingevuld worden.

Na normalisatie tot de derde normaalvorm ontstaan de volgende tabellen:

```
Hotel (id_hotel, name, street, number, id_city, stars)
Room (id_room, id_hotel, price_per_night, person_amount,
    nights_booked_this_year, date_last_booked)
Customer (customer_id, surname, name, birthdate, sex)
Booking (id_booking, id_room, id_hotel, id_customer, date_arrival,
    night_amount, paid)
Region (id_region, name, id_capital_city)
City (id_city, name, id_region)
```

Gevraagd:

1. Geef voor elke tabel de **primary key**?

2. Geef voor elke tabel de **foreign keys**?

3. Teken het **datastructuurdiagram** inclusief **cardinaliteiten** en **namen van relaties**.

4. Bespreek hoe de hotelid's kunnen bepaald worden van de hotels waarvoor nog geen enkele boeking (en/of optie) bestaat. Maak hierbij gebruik van verzamelings- en/of relationele operatoren.

5. Stel u voor dat een hotel failliet gaat (geschrapt moet worden). Bespreek gedetailleerd hoe u te werk gaat om alle informatie betreffende dit hotel uit de relationele database te verwijderen.

2.5.10 De eenvoudige bibliotheek

Ontwerp een sterk vereenvoudig relationeel databasemodel voor het beheer van een bibliotheek.

Van een boek wordt de titel, de auteur, het ISBN-nummer en een exemplaarnummer bijgehouden. In de situatie dat er bijvoorbeeld 3 exemplaren van hetzelfde boek aanwezig zijn, gebruikt men de exemplaarnummers: 1, 2 en 3. Verder houden we ook de status van het boek bij: ofwel is het boek *aanwezig* ofwel is het *uitgeleend*. De status wordt bijgehouden als een integer die respectievelijk 0 of 1 is.

Van een lid houden we bij: naam, lidnr, adres, gemeente.

Van elke ontlening houden we bij wie de ontlener is van welk boek en op welke dag het boek ontleend/teruggebracht werd. De informatie van alle ontleningen uit het verleden blijft behouden.

Gevraagd:

1. Geef het tupelschema van de nodige tabellen, duid de primary en foreign keys aan overeenkomstig de notatie in de cursus.

2. Teken het datastructuurdiagram, inclusief cardinaliteiten en namen van de relaties overeenkomstig de notatie in de cursus.

3. Bespreek hoe de ISBN-nummers kunnen bepaald worden van de boeken waarvan minstens één exemplaar aangekocht werd door de bibliotheek. Maak hierbij gebruik van verzamelings- en/of relationele operatoren.

4. Bespreek hoe de ISBN-nummers kunnen bepaald worden van de boeken waarvan nog nooit een exemplaar uitgeleend werd. Maak hierbij gebruik van verzamelings- en/of relationele operatoren.

5. Geef een lijst met ISBN-nummers en exemplaarnummers van alle boeken die momenteel uitgeleend zijn. Gebruik relationele algebra.

6. Stel u voor dat alle exemplaren van een boek uit de bibliotheek verwijderd worden. Bespreek gedetailleerd hoe u tewerk gaat om alle informatie betreffende dit boek uit de relationele database te verwijderen.

Het visueel weergeven van de database doen we met een ERD. We gebruiken hiervoor de tool Visual Paradigm.

De Powerpointpresentatie voor dit hoofdstuk vind je op Toledo.

Op Toledo vind je extra websites en filmpjes.