De Principes van Relationele Datamodellering: Een Theoretische en Praktische Gids

Deel 1: De Fundamenten van Relationele Datamodellering

1.1 De Essentie: Van Data naar Informatie en Databases

Voordat het ontwerp van een datamodel kan beginnen, is een helder begrip van de basisconcepten essentieel.

De concepten 'data' en 'informatie' worden vaak door elkaar gebruikt, maar vertegenwoordigen fundamenteel verschillende niveaus van abstractie.

- **Data** zijn de ruwe, ongeorganiseerde feiten die worden verzameld. Dit kunnen cijfers, tekst of andere observaties zijn die op zichzelf geen interpretatie of context bieden. Een voorbeeld is een lange lijst van productprijzen die wekelijks wordt genoteerd.
- **Informatie** ontstaat wanneer data wordt verwerkt, georganiseerd, gestructureerd en gepresenteerd binnen een specifieke context om het nuttig te maken. Informatie is geïnterpreteerde data. In het voorbeeld van de productprijzen zou de informatie de conclusie kunnen zijn dat de gemiddelde prijs van een product het afgelopen jaar is gestegen.

De brug tussen ruwe data en bruikbare informatie wordt vaak geslagen door een **database**. Een database (of databank) wordt gedefinieerd als een verzameling van permanente (of persistente) gegevens die zijn opgeslagen in een logische structuur. Deze verzameling gegevens wordt beheerd door een softwarepakket dat bekend staat als een **Database Management System (DBMS)**.[1]

1.2 Het Relationele Model: Eigenschappen van Tabellen, Rijen en Kolommen

Het meest gebruikte model voor het structureren van databases is het relationele model, geïntroduceerd door E.F. Codd in 1970. In dit model worden gegevens opgeslagen in tabellen, die de conceptuele basis vormen van de structuur.[1]

Een **tabel** bestaat uit:

- **Rijen** (ook wel records genoemd): Elke rij vertegenwoordigt één uniek exemplaar van het object dat de tabel beschrijft (bv. één specifieke klant, één specifieke bestelling).[1]
- **Kolommen** (ook wel attributen of velden genoemd): Elke kolom beschrijft een specifieke eigenschap van dat object (bv. de voornaam, het adres, de besteldatum).[1]

Het relationele model is niet zomaar een willekeurige spreadsheet; het wordt beheerst door strikte, fundamentele eigenschappen die de data-integriteit waarborgen [1]:

1. De volgorde van de rijen in een tabel is onbelangrijk.
2. De volgorde van de kolommen in een tabel is onbelangrijk.
3. Elke waarde in een specifieke kolom moet van hetzelfde type (datatype) zijn.
4. De kruising van een specifieke rij en een specifieke kolom kan maar één waarde bevatten (atomische waarden). Dit principe verklaart waarom samengestelde gegevens, zoals het opslaan van "Jan Peeters" in één veld, problematisch zijn.
5. **Cruciale Eigenschap:** Identieke rijen mogen niet voorkomen in een tabel.

Deze regel vormt de fundamentele rechtvaardiging voor het bestaan van sleutels: als geen twee rijen identiek mogen zijn, moet er een mechanisme bestaan om elke rij uniek te identificeren. Dit leidt direct tot het concept van de Primaire Sleutel.[1]

1.3 De Hiërarchie van Datamodellen: Van Concept tot Implementatie

Een datamodel is het resultaat van een gegevensanalyse en dient als de blauwdruk voor de database. Het is een cruciaal communicatiemiddel tussen de business (die de eisen definieert) en de ontwikkelaars (die de database bouwen). Dit proces verloopt in drie fasen van toenemende detaillering, die een brug slaan van abstracte business-semantiek naar concrete fysieke implementatie.[1]

1.3.0 Het Theoretische Fundament: Codd's Relationele Model

De basis voor moderne datamodellering werd in 1969-1970 gelegd door Edgar F. Codd.[4] Codd's doel was om de data-opslag *onafhankelijk* te maken van de fysieke hardware-implementatie en de applicaties die de data benaderden.[5] Zijn relationele model is een wiskundige benadering, gebaseerd op verzamelingenleer (set theory), die data organiseert in tabellen (of "relaties").[4]

Dit theoretische raamwerk vormt de kern van het **Logische Model (LDM)**. Het LDM is in essentie de praktische toepassing van Codd's relationele theorie, waarbij de focus ligt op een pure, genormaliseerde structuur van tabellen en sleutels, onafhankelijk van een specifieke softwareleverancier.[7]

1.3.1 Het Conceptuele Model (CDM)

Het conceptuele model (CDM) is het hoogste abstractieniveau. Het beschrijft *wat* er in het systeem moet worden opgeslagen en focust op de kernentiteiten en hun onderlinge verbanden, zoals begrepen door de business.[1]

- **Detailniveau:** Zeer abstract. Het dient primair als een *communicatiemiddel* tussen data-architecten en business-stakeholders (bv. managers, analisten) om te verzekeren dat de *semantiek* (de betekenis) van de business correct is vastgelegd.[2] Het bevat geen attributen (of slechts de hoogstnoodzakelijke), geen datatypes en geen sleutels.
- **Relaties:** Veel-op-veel ($M:N$) relaties zijn in deze fase toegestaan en zelfs gewenst, omdat ze de bedrijfsrealiteit vaak direct weerspiegelen (bv. "een auteur kan veel boeken schrijven, en een boek kan veel auteurs hebben").[1] Technische oplossingen voor deze relaties worden bewust weggelaten om de complexiteit laag te houden.

1.3.2 Het Logische Model (LDM)

Het logische model (LDM) vertaalt het abstracte conceptuele model naar een concrete, maar database-onafhankelijke structuur. Het beschrijft *hoe* de data gestructureerd wordt, gebaseerd op de principes van het relationele model.[1]

- **Detailniveau:** Dit is de *architecturale blauwdruk*. Het bevat alle entiteiten en hun attributen. Sleutels (PK, FK) worden aangeduid om de relaties te formaliseren.[1]
- **Datatypes:** Er worden algemene, DBMS-onafhankelijke datatypes gebruikt, zoals int, $float$, $string$, $date$ of $boolean$.[1]
- **Relaties:** Veel-op-veel ($M:N$) relaties zijn in deze fase niet langer toegestaan. Ze *moeten* worden opgelost (genormaliseerd) door ze te vervangen door associatietabellen (tussentabellen) en twee $1:M$ relaties.[1] Dit is een directe toepassing van Codd's normalisatieregels.

1.3.3 Het Fysieke Model (PDM)

Het fysieke model (PDM) is de definitieve blauwdruk voor de implementatie en is specifiek voor één gekozen Relationeel Database Management Systeem (RDBMS), zoals Oracle, MySQL of SQL Server.[1]

- **Detailniveau:** Dit model is de *implementatiegids* en is volledig uitgewerkt. Het vertaalt de logische architectuur naar een concrete, geoptimaliseerde implementatie.[3]
- **Sleutels:** Alle Primaire Sleutels (PK), Alternatieve Sleutels (AK) en Foreign Keys (FK) zijn gedefinieerd.
- **Datatypes:** Er worden specifieke datatypes gebruikt die horen bij het RDBMS (bv. $varchar(200)$ en $datetime2$ in SQL Server, versus $varchar2(200)$ en $date$ in Oracle).[1]
- **Constraints:** Alle beperkingen, zoals Null Allowed (NA) / Nulls Not Allowed (NNA) en de On delete specificaties (DTC, DTR, DTN), zijn hier vastgelegd.[1]
- **Performance:** Dit model bevat ook fysieke objecten die de performance beïnvloeden, zoals indexen. Soms wordt hier bewust *gedenormaliseerd* (afgeweken van het LDM) om de snelheid van veelvoorkomende queries te verhogen.[2]

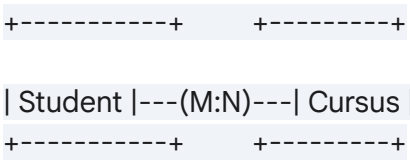
1.3.4 Visuele Progressie van de Modellen

Om de evolutie van een datamodel te illustreren, volgt hier een scenario: "Studenten schrijven zich in voor Cursussen".

Visueel Voorbeeld 1: Het Conceptuele Model (CDM)

Het CDM focust op de business-semantiek. Het model is eenvoudig en direct leesbaar voor een administrator of decaan.

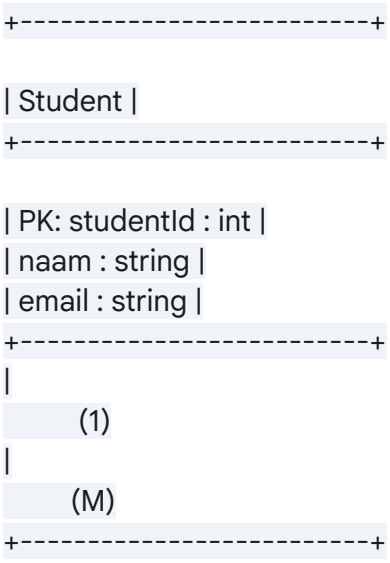
- Een entiteit Student.
- Een entiteit Cursus.
- Een $M:N$ (veel-op-veel) relatie "schrijft in" verbindt de twee.



Visueel Voorbeeld 2: Het Logische Model (LDM)

Het LDM lost de $M:N$ relatie op en definieert de structuur. Dit is de genormaliseerde, database-onafhankelijke blauwdruk.

- De $M:N$ relatie is opgesplitst door een nieuwe associatie-entiteit Inschrijving.
- Er zijn nu twee $1:M$ (één-op-veel) relaties.
- Logische attributen en sleutels zijn gedefinieerd.



```
| PK,FK: studentId : int |
| PK,FK: cursusId : int |
| inschrijfDatum : date|
+-------------------------+
        (M)
|
         (1)
|
+-------------------------+

| Cursus |
+-------------------------+

| PK: cursusId : int |
| titel : string |
| studiepunten : int |
+-------------------------+
```

Visueel Voorbeeld 3: Het Fysieke Model (PDM) (Voorbeeld: MS SQL Server)

Het PDM is de concrete implementatiegids, specifiek voor een RDBMS (hier MS SQL Server). Het bevat alle technische details, inclusief constraints die in Deel 5 worden besproken.

```
+-----------------------------------------------------+

| Student (Tabel) |
+-----------------------------------------------------+

| studentId : int IDENTITY(1,1) {PK, NNA} |
| naam : varchar(100) {NNA} |
| email : varchar(255) {NNA, AK_Student_Email UNIQUE} |
+-----------------------------------------------------+
|
         (1)
|
        (M) {FK_Inschrijving_Student, ON DELETE CASCADE}
+-----------------------------------------------------+

| Inschrijving (Tabel) |
```

```
+-----------------------------------------------------+

| studentId : int {PK, FK, NNA} |
| cursusId : int {PK, FK, NNA} |
| inschrijfDatum : datetime2 {NNA, DEFAULT GETDATE()} |
+-----------------------------------------------------+
        (M) {FK_Inschrijving_Cursus, ON DELETE CASCADE}
|
         (1)
|
+-----------------------------------------------------+

| Cursus (Tabel) |
+-----------------------------------------------------+

| cursusId : int IDENTITY(1,1) {PK, NNA} |
| titel : varchar(100) {NNA} |
| studiepunten : int {NNA, CK_Studiepunten CHECK > 0} |
+-----------------------------------------------------+
```

1.4 Vergelijkende Analyse: Conceptueel, Logisch en Fysiek

De progressieve evolutie van een datamodel, van abstract idee tot concrete implementatie, kan worden samengevat in de volgende tabel. Deze tabel is gebaseerd op de analyse in de verstrekte documenten en is van cruciaal belang voor het begrip van het modelleringsproces.[1]

ERD onderdeel	Conceptueel	Logisch	Fysiek
Entiteit (tabel) naam	☑	☑	☑
Relaties	☑	☑	☑
Veel op veel toegestaan	☑	Optioneel (meestal niet)	X
Kolom (attribuut)	X (of zeer beperkt)	☑	☑

naam			
Datatypes	X	Optioneel (algemeen)	☑ (specifiek)
Primary key (PK)	X	Optioneel	☑
Foreign key (FK)	X	Optioneel	☑
Relatiespecificaties (NA/NNA, On Delete)	X	Optioneel	☑

Deel 2: De Bouwstenen van het ERD: Entiteiten en Attributen

2.1 Entiteiten: Definitie, Regels en Naamgevingsconventies

Een **entiteit** is het centrale concept in een ERD. Het vertegenwoordigt een object (een persoon, plaats, ding, concept of gebeurtenis) waarover de organisatie informatie wenst te bewaren. Een fundamentele eis is dat elke entiteit (en elk record daarbinnen) uniek identificeerbaar moet zijn.[1]

Naamgevingsconventie (Regel):

- De naam van een entiteit begint met een **H**oofdletter.
- De naam staat altijd in het **E**nkelvoud.
- Voorbeeld: Klant, Product, Order (niet Klanten, Producten, Orders).[1]

2.2 Attributen: Definitie, Regels en Naamgevingsconventies

Een **attribuut** is een kenmerk of een specifieke eigenschap van een entiteit. Het beschrijft een stukje data dat we van de entiteit willen bijhouden.

Naamgevingsconventie (Regel):

- De naam van een attribuut begint met een **k**leine letter.
- Indien de naam uit meerdere woorden bestaat, wordt **c**amel**C**ase gebruikt (het tweede en elk volgend woord beginnen met een hoofdletter).
- De naam staat in het enkelvoud en moet duidelijk zijn (geen afkortingen).
- Voorbeeld: voornaam, straat, leveringsdatum, commissiePercentage.

Conventie Primaire Sleutel (Regel):

- De primaire sleutel, die dient als de unieke technische identifier, krijgt binnen deze conventie altijd de naam id.[1]

2.3 Essentiële 'Best Practices' voor Attribuutontwerp

Een slecht ontworpen attribuutstructuur kan leiden tot data-redundantie, update-anomalieën en onmogelijke queries. De volgende regels zijn cruciaal voor een robuust model.

2.3.1 Het Vermijden van Samengestelde Gegevens (Composite Data)

Dit zijn attributen die logisch opgesplitst kunnen worden in meerdere, kleinere onderdelen, zoals een 'volledige naam' of een 'adres'.[1]

- **Probleem:** Het opslaan van "Jan Peeters" in een enkel naam veld maakt het extreem moeilijk of onmogelijk om de data betrouwbaar te gebruiken. Een applicatie kan niet eenvoudig een gepersonaliseerde e-mail sturen ("Beste Jan"). Het wordt nog complexer bij namen met tussenvoegsels ("Jan-Willem van de Velde").
- **Regel:** Bewaar samengestelde gegevens altijd in hun kleinste, logische, atomische componenten.
    - naam $\rightarrow$ voornaam en naam.[1]
    - adres $\rightarrow$ straat, huisnummer, postcode, gemeente.[1]

2.3.2 Het Vermijden van Proces- of Berekende Gegevens (Calculated Data)

Dit zijn gegevens die afgeleid of berekend kunnen worden op basis van andere attributen die al in de database aanwezig zijn. Het klassieke voorbeeld is leeftijd.

- **Probleem:**
    1. **Inefficiëntie en Onderhoud:** De leeftijd van elke persoon in de database verandert elk jaar. Dit zou vereisen dat het systeem dagelijks of jaarlijks elk record moet bijwerken (updaten), wat een "compleet onnodige" en zware systeembelasting is.[1]
    2. **Data-Integriteit:** Het creëert een risico op inconsistentie. Wat als de geboortedatum 01-01-1990 is, maar het leeftijd veld (door een fout) nog op 33 staat? Welk veld is dan de waarheid?
- **Regel:** Bewaar nooit berekende gegevens. Bewaar altijd de bron van de berekening (in dit geval geboortedatum). De applicatie (of een database view) kan de leeftijd vervolgens 'on-the-fly' berekenen wanneer dat nodig is.

2.3.3 Correct Gebruik van Datatypes

De keuze van een datatype moet de semantiek van de data weerspiegelen, niet alleen het uiterlijk.

- **Regel:** "Gebruik enkel getallen als je wil rekenen".
- **Voorbeeld:** Een Postcode (bv. 2440) en een Telefoonnummer zijn geen getallen. Er wordt nooit mee gerekend (de som van twee postcodes is zinloos). Het zijn $string$ (tekst) types. Een Salaris is daarentegen wel een getal (bv. $float$), omdat hiermee gerekend moet worden.[1]

2.4 Soorten Entiteiten Onderscheiden

Entiteiten kunnen worden gecategoriseerd op basis van hun rol en afhankelijkheid binnen het model.

2.4.1 Kernentiteiten (Core Entities)

Dit zijn de fundamentele, onafhankelijke entiteiten in het model. Ze "bestaan op zichzelf" en hebben geen andere entiteit nodig om hun betekenis te krijgen. Dit zijn typisch de hoofdonderwerpen van de applicatie.

- **Voorbeeld:** Klant, Product, Leverancier, Speler.

2.4.2 Associatie-entiteiten (Association Entities)

Een associatie-entiteit (of tussentabel) bestaat niet op zichzelf, maar dient als een 'brug' om een veel-op-veel ($M:N$) relatie tussen twee of meer andere (meestal kern)entiteiten weer te geven.

- **Voorbeeld:** Een bestelling kan veel producten bevatten, en een product kan in veel bestellingen voorkomen. De entiteit Orderlijn (of BestellingProduct) is de associatie-entiteit die Order en Product koppelt. Een ander voorbeeld is BookAuthor die Book en Author koppelt.[1]

2.4.3 Karakteristieke Entiteiten (Characteristic Entities)

Een karakteristieke entiteit bestaat ook niet op zichzelf, maar dient uitsluitend om een andere (meestal kern)entiteit te verduidelijken, kwalificeren of te karakteriseren.[1]

- **Doel:** Het wordt gebruikt om een lijst van waarden (zoals types, categorieën of statussen) te beheren die dynamisch is (de lijst kan groeien of wijzigen) of waaraan extra eigenschappen gekoppeld moeten worden.[1]
- **Voorbeeld:** Een Auto heeft een merk. In plaats van merk als een $string$ attribuut op te slaan (wat kan leiden tot inconsistente invoer zoals "VW", "vW", "volkswagen"), wordt een aparte karakteristieke entiteit Merk aangemaakt. De Auto entiteit krijgt dan een Foreign Key (FK) die verwijst naar de Merk entiteit.[1]

Deel 3: De Architectuur van Data: Sleutels (Keys)

Sleutels zijn de mechanismen die de relationele regels (zoals uniciteit en verbanden)

afdwingen. Het selecteren van de juiste sleutels is geen willekeurige keuze, maar volgt een logisch proces van eliminatie en selectie.

3.0 De Sleutelhiërarchie: Een Proces van Selectie

Om de verschillende soorten sleutels te begrijpen, moeten we ze zien als een hiërarchische trechter. Men begint met een brede set van mogelijkheden, die vervolgens wordt verfijnd tot de uiteindelijke sleutels.[10]

3.0.1 Superkey (Supersleutel)

- **Definitie:** Een *Superkey* (supersleutel) is de breedste categorie. Het is *elke* set van één of meer attributen die, wanneer gecombineerd, een rij in een tabel uniek kan identificeren.[10]
- **Kenmerk:** Een supersleutel mag "overbodige" (extraneous) attributen bevatten.[10]
- **Voorbeeld:** Beschouw een tabel Student met de kolommen {id, studentenNummer, email, naam}. Aangenomen dat id, studentenNummer en email allemaal uniek zijn, zijn de volgende combinaties (en vele andere) allemaal *Superkeys*:
  - {id}
  - {studentenNummer}
  - {id, naam}
  - {studentenNummer, email, naam}

3.0.2 Candidate Key (Kandidaatsleutel)

- **Definitie:** Een *Candidate Key* (kandidaatsleutel) is een *minimale supersleutel*. Dit betekent dat het nog steeds uniek is, maar er kunnen geen attributen meer uit de set worden verwijderd zonder dat de uniciteit verloren gaat.[10] Dit zijn de *echte* kandidaten om als Primaire Sleutel te dienen.
- **Kenmerk:** Een tabel kan *meerdere* kandidaatsleutels hebben.[12]
- **Voorbeeld:** In het Student voorbeeld hierboven zijn de *Candidate Keys*:
  - {id}
  - {studentenNummer}
  - {email}
  - De combinatie {id, naam} is *geen* kandidaatsleutel, omdat het een *superkey* is waaruit

naam kan worden verwijderd terwijl de uniciteit (gegarandeerd door id) behouden blijft.[10]

3.1 De Primaire Sleutel (PK)

De **Primaire Sleutel (PK)** is de *gekozen* kolom, of een combinatie van kolommen, die één specifieke rij (record) in een tabel uniek identificeert.[1]

- **Relatie tot Kandidaatsleutel:** De Primaire Sleutel is de *éne* kandidaatsleutel die door de databaseontwerper is *geselecteerd* om als de officiële, primaire identifier voor de tabel te dienen.[10]

De PK moet voldoen aan twee strikte integriteitsregels:

1. **Uniciteit:** De waarde van de PK mag nooit twee keer voorkomen in de tabel.
2. **Verplicht:** De waarde van de PK mag nooit $NULL$ (leeg) zijn. In een fysiek model wordt dit genoteerd als {NNA (Nulls Not Allowed)}.[1]

3.1.5 Het Verschil: Primary Key vs. UNIQUE Constraint

Een veelvoorkomende verwarring ontstaat tussen de Primaire Sleutel en een UNIQUE constraint (die gebruikt wordt voor Alternatieve Sleutels, zie 3.3). Hoewel beide uniciteit afdwingen, zijn er fundamentele verschillen:

- **Nullability:** Een Primaire Sleutel dwingt een NOT NULL constraint af; $NULL$ is per definitie niet toegestaan.[1] Een UNIQUE constraint staat in de meeste RDBMS'en wel $NULL$-waarden toe (vaak slechts één, aangezien $NULL$ niet gelijk wordt geacht aan een andere $NULL$).[15]
- **Aantal per Tabel:** Een tabel kan slechts *één* Primaire Sleutel hebben.[15] Een tabel kan *meerdere* UNIQUE constraints hebben (bv. één op email en één op telefoonnummer).[17]
- **Doel:** De PK definieert de *identiteit* van de rij (entity integrity).[16] Een UNIQUE constraint dwingt een specifieke *business rule* af die uniciteit vereist voor een niet-primaire kolom.[16]
- **Fysieke Implementatie:** In veel RDBMS'en (zoals SQL Server) wordt de PK standaard geïmplementeerd als een *clustered index* (die de fysieke opslagvolgorde van de data op schijf bepaalt), terwijl een UNIQUE constraint standaard als een *non-clustered index* wordt geïmplementeerd.[17]

3.2 De Surrogaat Sleutel (Surrogate Key): De 'Best Practice'

Er zijn twee soorten sleutels die als PK kunnen dienen:

1. **Natuurlijke Sleutel:** Een attribuut dat in de 'echte wereld' al uniek is (bv. studentenNummer, factuurnummer, rijksregisternummer).[1]
2. **Surrogaat Sleutel (Synthetische Sleutel):** Een sleutel die geen betekenis heeft in de 'echte wereld', maar puur technisch wordt toegevoegd door de databaseontwerper om als unieke identifier te dienen.[1]

Binnen de context van deze cursus en als algemene 'best practice' wordt de voorkeur gegeven aan de surrogaat sleutel. Deze wordt standaard id genoemd en is van het type $integer$.[1]

De voorkeur voor een surrogaat sleutel is gebaseerd op vier essentiële voordelen [1]:

1. **Heeft geen betekenis:** Dit zorgt voor stabiliteit. Stel dat een studentenNummer (een natuurlijke sleutel) als PK wordt gebruikt en bij invoer wordt een typfout gemaakt. Later, wanneer de student met het echte nummer zich probeert in te schrijven, faalt het systeem. Een betekenisloze id heeft dit probleem niet.
2. **Is minimaal:** Een $integer$ (bv. 4 bytes) is veel kleiner dan een $string$ (bv. $varchar(50)$). Bij het leggen van verbanden (joins) op grote tabellen resulteert een kleinere sleutel in significant snellere operaties en minder schijfgebruik.
3. **Wordt nooit geüpdatet:** Een natuurlijke sleutel kan wijzigen (een persoon verandert van e-mailadres, een bedrijf verandert van BTW-nummer). Als dit de PK is, moet deze wijziging ook worden doorgevoerd in alle tabellen die ernaar verwijzen (via Foreign Keys). Dit is een zware en risicovolle operatie. Een surrogaat id wijzigt nooit.[1]
4. **Is een integer:** Vergelijkingen tussen getallen (int) zijn voor een database aanzienlijk sneller en efficiënter dan vergelijkingen tussen teksten ($string$).[1]

3.3 Alternatieve Sleutels (AK) (Alternative Keys)

Wanneer een surrogaat id als PK is gekozen, wat gebeurt er dan met de natuurlijke sleutels? Dit worden **Alternatieve Sleutels (AK)**.[1]

- **Definitie:** Een Alternatieve Sleutel is een *elke kandidaatsleutel die niet werd gekozen als de Primaire Sleutel*.[13]

- **Doel:** Hoewel de id (PK) de *technische* uniciteit garandeert, garandeert de AK de *business* uniciteit. Het zorgt ervoor dat geen twee studenten hetzelfde studentenNummer of email kunnen hebben.[1]
- **Implementatie:** Een AK krijgt in SQL (meestal) een UNIQUE constraint en wordt vaak geïndexeerd om snel zoeken op deze kolom mogelijk te maken.[1]
- **ERD Regel:** Alle kernentiteiten moeten (minimaal) één AK hebben.

3.4 De Foreign Key (FK)

De **Foreign Key (FK)** (of refererende sleutel) is de 'lijm' van de relationele database.

- **Definitie:** Een FK is een kolom (of combinatie van kolommen) in één tabel (de 'child' tabel) die verwijst naar de Primaire Sleutel (PK) in een andere tabel (de 'parent' tabel).[1] (Een FK kan technisch ook verwijzen naar een Alternatieve Sleutel / UNIQUE constraint).
- **Doel:** Het legt het verband (de relatie) vast tussen twee tabellen en voorkomt data duplicatie. In plaats van de volledige naam en details van een serie te herhalen voor elk boek, slaat de Book tabel enkel de id van de serie op als een FK.[1]

3.5 Samengestelde Sleutels (Composite Keys)

- **Definitie:** Een samengestelde sleutel (of *composite key*) is een sleutel (PK, AK, of FK) die bestaat uit een combinatie van *twee of meer* kolommen om uniciteit te garanderen.[23]
- **Toepassing 1: Associatie-entiteiten (M:N Relaties):** Dit is de meest voorkomende toepassing. Bij het oplossen van een $M:N$ relatie tussen Student en Cursus, wordt de PK van de tussentabel Inschrijving vaak gevormd door de combinatie van de twee FK's: {FK_studentId, FK_cursusId}.[23] Dit garandeert dat een student zich maar één keer voor dezelfde cursus kan inschrijven.
- **Toepassing 2: Zwakke Entiteiten (Identificerende Relaties):** Zoals besproken in Deel 4.5.3, kan de PK van een 'kind'-entiteit bestaan uit de PK van de 'ouder' plus een eigen discriminator. Bijvoorbeeld, de PK van een Kamer kan zijn {FK_gebouwId, kamerNummer}.
- **SQL Voorbeeld (Composite PK):** Het volgende voorbeeld toont de creatie van een OrderLijn tabel (een associatie-entiteit) waarbij de PK is samengesteld uit orderId en productId.[23]

SQL

```
CREATE TABLE OrderLijn (
    orderId INT NOT NULL,
    productId INT NOT NULL,
    aantal INT NOT NULL,

    -- Definitie van de samengestelde Primaire Sleutel
    CONSTRAINT PK_OrderLijn PRIMARY KEY (orderId, productId),

    -- Definitie van de individuele Foreign Keys
    CONSTRAINT FK_OrderLijn_Order
        FOREIGN KEY (orderId) REFERENCES Order(id),
    CONSTRAINT FK_OrderLijn_Product
        FOREIGN KEY (productId) REFERENCES Product(id)
);
```

3.6 Samenvattende Tabel: De Sleutelhiërarchie

De relatie tussen de verschillende sleuteltypes kan als volgt worden samengevat:

Sleuteltype	**Definitie**	**Voorbeeld (Tabel: Student)**	**Aantal per Tabel**	**NULL Toegestaan?**
Superkey	*Elke* set attributen die een rij uniek identificeert.	{id, naam}	Vele	Ja
Candidate Key	Een *minimale* superkey (geen overbodige attributen).	{id}, {email}	Eén of meer	Nee
Primary Key	De *gekozen*	{id}	Precies één	Nee [1]

(PK)	kandidaatsleutel.			
Alternate Key (AK)	Een kandidaatsleutel die *niet* als PK is gekozen.	{email}	Nul of meer	Ja (meestal 1x) [16]
Surrogate Key (SK)	Een technisch, betekenisloos $integer$ veld, gebruikt als PK.	{id} (als auto-nummer)	Nul of één	Nee
Composite Key	Een sleutel (PK, AK, etc.) die uit 2+ kolommen bestaat.	{voornaam, naam, geboortedatum}	N.v.t.	Afhankelijk van type
Foreign Key (FK)	Een kolom die verwijst naar de PK (of AK) van een andere tabel.	{FK_opleidingId}	Nul of meer	Ja (optioneel) / Nee (verplicht)

Deel 4: Het Modelleren van Verbanden: Relaties en Multipliciteit

4.1 De Drie Soorten Relaties (Multipliciteit)

Multipliciteit (ook wel cardinaliteit genoemd) beschrijft hoeveel records van de ene entiteit kunnen worden gekoppeld aan hoeveel records van een andere entiteit.

1. **Eén-op-één (1:1):** Elk record in A heeft precies één gerelateerd record in B.[25]
    - **Toepassing:** Zeer zeldzaam. Het is vaak een indicatie dat entiteit A en B samengevoegd moeten worden tot één entiteit. Soms gebruikt voor performance of beveiliging (bv. Werknemer en WerknemerGeheim met $1:1$ relatie voor salarisdata).[25]
2. **Eén-op-veel (1:M of 1:*):** Elk record in A kan meerdere gerelateerde records in B hebben, maar elk record in B heeft slechts één gerelateerd record in A.[27]
    - **Toepassing:** Dit is de meest voorkomende en fundamentele relatie (bv. één Klant heeft veel Bestellingen).[27]
3. **Veel-op-veel (M:N of *:*):** Elk record in A kan meerdere records in B hebben, en vice versa.[28]
    - **Toepassing:** Dit kan niet direct geïmplementeerd worden in een relationele database en vereist een specifieke oplossing.[1]

4.2 De Gouden Regel: Correcte Plaatsing van de Foreign Key

De plaatsing van de FK in een $1:M$ relatie is geen keuze, maar een logische noodzaak om data-redundantie te voorkomen. Dit wordt geïllustreerd met de $1:M$ relatie tussen Series (1) en Book (M).[1]

- **Optie 1 (Fout):** Plaats de FK bookId in de Series tabel (de '1'-kant).
    - **Resultaat:** Als "The Kharkanas Trilogy" (id 1) twee boeken heeft (id 1 en 2), moet de Series tabel twee keer worden aangemaakt: één record voor (Series 1, Book 1) en één record voor (Series 1, Book 2). Dit dupliceert de serie-informatie ("The Kharkanas Trilogy") onnodig.
- **Optie 2 (Correct):** Plaats de FK seriesId in de Book tabel (de 'veel'-kant).
    - **Resultaat:** De Series tabel heeft één record: (id 1, "The Kharkanas Trilogy"). De Book tabel kan nu meerdere records hebben die allemaal verwijzen naar seriesId = 1. Er is geen duplicatie van data.[1]

REGEL: "De FK staat altijd aan de * kant (de 'veel'-kant) van elke relatie".[1]

4.3 De Oplossing voor M:N Relaties: De Associatie-entiteit

Het bovenstaande probleem wordt complexer bij een $M:N$ relatie, zoals Book (*:*) Author.

- **Probleem:** Waar komt de FK?

    - Als de FK authorId in Book staat (Optie 2 van $M:N$), moet een boek dat door twee auteurs is geschreven, twee keer in de Book tabel worden opgeslagen (duplicatie van boeken).
    - Als de FK bookId in Author staat (Optie 1 van $M:N$), moet een auteur die twee boeken heeft geschreven, twee keer in de Author tabel worden opgeslagen (duplicatie van auteurs).
    - Beide opties leiden tot onwerkbare data-redundantie.[1]
- **Oplossing:** De $M:N$ relatie wordt opgesplitst met behulp van een derde entiteit: de **associatie-entiteit** (of tussentabel/samenvoegtabel), bijvoorbeeld BookAuthor.[1]
- **Mechanisme:** Deze tussentabel lost de $M:N$ relatie op door deze te vervangen door twee $1:M$ relaties [28]:
    1. Book (1) $\rightarrow$ BookAuthor (M)
    2. Author (1) $\rightarrow$ BookAuthor (M)
- De associatietabel BookAuthor bevat (minimaal) twee FK's: FK1 bookId en FK2 authorId. Een record in deze tabel (bv. bookId = 6, authorId = 2) koppelt één specifiek boek aan één specifieke auteur, zonder enige data in de Book of Author tabellen te dupliceren.[1]

4.4 Multipliciteit in Detail: Aard en Connectiviteit

De notatie van een relatie (bv. $0..*$ of $1..1$) bestaat uit twee afzonderlijke componenten die elk een specifieke regel definiëren.[1]

- **Aard (Deelname):** Dit is het *minimum* aantal keer dat een relatie *moet* voorkomen. Het bepaalt of de relatie **optioneel (0)** of **verplicht (1)** is.
    - **0 (Optioneel):** Een record *kan* bestaan zonder de relatie. Voorbeeld: In Klant (0).. Order (*), kan een Klant bestaan zonder dat deze al een Order heeft geplaatst.
    - **1 (Verplicht):** Een record *moet* de relatie hebben om te kunnen bestaan. Voorbeeld: In Order (1).. Klant (*), kan een Order niet bestaan zonder gekoppeld te zijn aan een Klant.[1]
- **Connectiviteit:** Dit is het *maximum* aantal keer dat een relatie *kan* voorkomen. Het bepaalt of de relatie **één (1)** of **veel (*)** is.

Deze twee componenten (Aard en Connectiviteit) zijn direct verbonden met de fysieke implementatie van de Foreign Key [1]:

- Als de **Aard** aan de kant van de FK **1 (verplicht)** is (zoals bij Order $\rightarrow$ Klant), moet de FK-kolom (klantId) ingesteld worden als **NNA (Nulls Not Allowed)**.
- Als de **Aard** aan de kant van de FK **0 (optioneel)** is (zoals bij Book $\rightarrow$ Series, een boek hoeft niet tot een serie te behoren), moet de FK-kolom (seriesId) ingesteld worden als **NA (Nulls Allowed)**.

4.5 Taxonomie van Gevorderde Relatietypes

Naast de standaard binaire relaties (tussen twee entiteiten) bestaan er complexere structuren die specifieke modelleringspatronen vereisen.

4.5.1 Unaire (Recursieve) Relaties

- **Definitie:** Een unaire of recursieve relatie is een relatie waarbij een entiteit een relatie heeft *met zichzelf*. Dit wordt ook een *self-referencing relationship* genoemd.[30]
- **Toepassing:** Dit patroon is essentieel voor het modelleren van hiërarchische of boomstructuren binnen één tabel.[30]
- **Voorbeeld (Werknemer Hiërarchie):** De meest klassieke toepassing is het organigram van een bedrijf.
  - Entiteit: Werknemer
  - Attributen: id (PK), naam, managerId (FK).
  - De Relatie: De kolom managerId is een Foreign Key die *verwijst naar de id-kolom in dezelfde Werknemer-tabel*.[30]
  - Een werknemer wiens managerId $NULL$ is, staat aan de top van de hiërarchie (bv. de CEO).
- **SQL Voorbeeld:**

SQL

```
CREATE TABLE Werknemer (
    id INT PRIMARY KEY,
    naam VARCHAR(100),
    -- De FK verwijst naar de PK van dezelfde tabel
    managerId INT NULL,

    CONSTRAINT FK_Werknemer_Manager FOREIGN KEY (managerId)
    REFERENCES Werknemer(id)
    -- ON DELETE SET NULL is hier vaak een goede keuze (zie 5.4)
);
```

Het bevragen van een dergelijke structuur (bv. "geef mij alle ondergeschikten van manager X") vereist speciale *recursieve queries* (vaak met Common Table Expressions - CTEs).[32]

4.5.2 Ternaire (en N-aire) Relaties

- **Definitie:** Een ternaire relatie is een relatie die *drie* verschillende entiteiten tegelijkertijd met elkaar verbindt.[36] Een N-aire relatie verbindt N entiteiten.
- **Probleem:** Een ternaire relatie kan *niet* worden opgesplitst in (bijvoorbeeld) drie afzonderlijke binaire relaties zonder dat de *context* of *semantiek* verloren gaat.
- **Voorbeeld (Folksonomy):** Een Gebruiker past een Tag toe op een Boek.[37]
    - Een record (Jan, SQL, Database Design) betekent dat *Jan* de tag *SQL* heeft toegepast op het boek *Database Design*.
    - Dit is niet hetzelfde als drie losse feiten: (Jan, SQL), (Jan, Database Design), (SQL, Database Design).
- **Oplossing:** Net als bij $M:N$ relaties wordt een associatie-entiteit gebruikt, maar deze entiteit bevat nu *drie* (of N) Foreign Keys.[37]
- **Structuur:**
    - Entiteiten: Gebruiker, Tag, Boek.
    - Associatie-entiteit: GebruikerTagBoek
    - Sleutels in GebruikerTagBoek:
        - FK_gebruikerId (verwijst naar Gebruiker)
        - FK_tagId (verwijst naar Tag)
        - FK_boekId (verwijst naar Boek)
        - De Primaire Sleutel is typisch een samengestelde sleutel van alle drie de FK's: {FK_gebruikerId, FK_tagId, FK_boekId}.[37]

4.5.3 Identificerende vs. Niet-Identificerende Relaties

Dit onderscheid definieert hoe een 'kind'-entiteit (de 'veel'-kant) zich verhoudt tot zijn 'ouder'-entiteit (de 'één'-kant), specifiek met betrekking tot zijn *bestaansrecht* en *identificatie*.[38]

- **Niet-Identificerende Relatie (Meest Voorkomend):**
    - **Definitie:** Dit is de standaard $1:M$ relatie. De 'kind'-entiteit heeft een *eigen, onafhankelijke* Primaire Sleutel (meestal een surrogaat id).
    - **Sleutels:** De PK van de 'ouder' migreert naar het 'kind' als een *gewone Foreign Key*

die *geen* deel uitmaakt van de PK van het kind.[38]
    - **Voorbeeld:** Klant (PK: id) en Bestelling (PK: id, FK: klantId). De klantId in Bestelling is enkel een FK.
- **Identificerende Relatie (Relatie met een Zwakke Entiteit):**
    - **Definitie:** Wordt gebruikt wanneer de 'kind'-entiteit *logisch niet kan bestaan* zonder de 'ouder'.[38] Het 'kind' wordt in deze context een **Zwakke Entiteit** (Weak Entity) genoemd, omdat het zijn identiteit deels ontleent aan de 'ouder'.[39]
    - **Sleutels:** De PK van de 'ouder' migreert naar het 'kind' en wordt daar *onderdeel van de Primaire Sleutel van het kind*.[38]
    - **Voorbeeld:** Gebouw en Kamer. Een Kamer is betekenisloos zonder een Gebouw en wordt geïdentificeerd door zijn *kamernummer* (een partiële sleutel) *binnen* een specifiek gebouw.
        - Gebouw {PK: gebouwId}
        - Kamer {PK: (FK_gebouwId, kamerNummer)}
    - In dit geval is FK_gebouwId in de Kamer tabel *zowel* een Foreign Key *als* een deel van de Primaire Sleutel. Het gebruik van surrogaat id sleutels (zoals aanbevolen in 3.2) maakt dit patroon zeldzamer, omdat ontwerpers vaak toch kiezen voor een niet-identificerende relatie (bv. Kamer {PK: kamerId, FK: gebouwId}).[38]

4.6 Samenvattende Tabel: Relatietypes

Relatietype	Definitie	Klassiek Voorbeeld	Implementatie
Binair (1:M)	Eén 'ouder' record heeft veel 'kind' records.	Klant $\rightarrow$ Bestelling	FK in de 'veel'-tabel (Bestelling).
Binair (M:N)	Veel records hebben veel records.	Student $\leftrightarrow$ Cursus	Associatietabel met 2 FK's.
Unair (Recursief)	Een entiteit heeft een relatie met zichzelf.	Werknemer $\rightarrow$ Werknemer (Manager)	FK in dezelfde tabel, verwijst naar eigen PK.

Ternair (N-air)	Een relatie tussen drie (of meer) entiteiten.	Gebruiker past Tag toe op Boek	Associatietabel met 3 (of N) FK's.
Identificerend	Een 'zwak kind' dat niet kan bestaan zonder de 'ouder'.	Gebouw $\rightarrow$ Kamer	PK van 'ouder' wordt *deel van* de PK van 'kind'.

Deel 5: Gedetailleerde Specificatie en Integriteitsregels

In het Fysieke Model (PDM) wordt de logica van het LDM vertaald naar afdwingbare regels. Deze *constraints* vormen het "immuunsysteem" van de database; ze beschermen actief de data-integriteit door ongeldige data te weigeren.[40]

5.1 Datatypes: Logisch vs. Fysiek Niveau

De keuze van datatypes wordt specifieker naarmate het model evolueert.

- **Logisch Niveau:** Gebruik algemene, DBMS-onafhankelijke types: int, $float$, $string$, $date$, $time$, $datetime$, $boolean$.[1]
- **Fysiek Niveau:** Gebruik DBMS-specifieke types, zoals $varchar2$ en $number$ (Oracle) of $varchar$ en $datetime2$ (SQL Server).[1]
- **Notatie:** Datatypes worden in het ERD achter een dubbelpunt genoteerd: naam: string.

5.2 De Afweging: Karakteristieke Entiteit versus Enumeratie (Enum)

Voor attributen met een beperkte set van mogelijke waarden (bv. brandstoftype, merk, status) zijn er twee oplossingen: een enum of een karakteristieke entiteit.

- **Enumeratie (Enum):**
  - **Definitie:** Een vaste, stabiele lijst met waarden, gedefinieerd binnen het attribuut zelf.

    - **Structuur:** Er wordt geen aparte entiteit aangemaakt.[1]
    - **Beperking:** Er kunnen geen extra eigenschappen aan de waarden worden gekoppeld.
    - **Voorbeeld:** Het attribuut brandstofType met de vaste, onveranderlijke lijst: 'Benzine', 'Diesel', 'Hybride', 'Elektrisch'.[1]
- **Karakteristieke Entiteit:**
    - **Definitie:** Een aparte entiteit (tabel) die wordt aangemaakt om een dynamische lijst van waarden te beheren.
    - **Structuur:** De lijst kan groeien of wijzigen (bv. er komt een nieuw automerk bij) en er kunnen extra eigenschappen aan de waarden worden gekoppeld (bv. het land van herkomst van het merk).
    - **Voorbeeld:** Een Merk entiteit die via een FK gekoppeld is aan de Auto entiteit.
- **Vuistregel (Regel):** "gebruik een enum als de lijst klein en stabiel is"; "gebruik een karakteristieke entiteit als de waarden vaak wijzigen of als je er extra eigenschappen/relaties aan wilt koppelen".[1]

5.3 Null-Specificaties: Nulls Allowed (NA) vs. Nulls Not Allowed (NNA)

Voor elk attribuut in het fysieke model moet worden aangegeven of het $NULL$ (leeg) mag zijn. Dit is de constraint die *Entity Integrity* (voor PK's) en *Domain Integrity* (voor gewone kolommen) bewaakt.

- **NA (Nulls Allowed):** Lege waarden zijn toegestaan. Dit is nuttig voor optionele data (bv. einddatum voor een bestuurslid dat nog in functie is).[1]
- **NNA (Nulls Not Allowed):** Het attribuut moet altijd een waarde hebben. Dit geldt voor verplichte data (bv. begindatum).[1]
- **Notatie:** Wordt genoteerd tussen accolades achter het datatype: begindatum: date {NNA}.[1]
- **Regel:** De Primaire Sleutel (PK) is altijd {NNA}.[1]

De Theoretische Complexiteit van NULL

De specificatie NA/NNA is meer dan een simpele keuze; het heeft diepgaande theoretische implicaties.

- **NULL is geen waarde:** $NULL$ is geen 0, geen lege string ("") en geen $false$. Het is een *marker* voor een "onbekende," "ontbrekende" of "niet-toepasselijke" waarde.[43]

- **Three-Valued Logic (3VL):** Omdat $NULL$ "onbekend" betekent, breekt het de standaard binaire logica (TRUE/FALSE). SQL gebruikt een drie-waarden-logica: TRUE, FALSE, en UNKNOWN.[45]
    - Een vergelijking zoals leeftijd = 25 is TRUE of FALSE.
    - Een vergelijking zoals leeftijd = NULL is *altijd* UNKNOWN (omdat we niet weten wat leeftijd is).
    - Een WHERE clausule retourneert enkel rijen waar de conditie TRUE is. Rijen die FALSE of UNKNOWN evalueren, worden *niet* geretourneerd. Dit is een veelvoorkomende bron van bugs.
- **Best Practice:** Om de complexiteit van 3VL te vermijden, is het vaak beter om kolommen als {NNA} (NOT NULL) te definiëren en een DEFAULT constraint (zie 5.5.3) te gebruiken om een bekende standaardwaarde (zoals 0 of 'Onbekend') in te vullen. Dit moet echter afgewogen worden tegen de business-semantiek: als "onbekend" een legitieme en aparte staat is, dan is $NULL$ de correcte keuze.[43]

5.4 Referentiële Integriteit: Foreign Key Specificaties bij Verwijdering (DTC/DTR/DTN)

Referentiële integriteit zorgt ervoor dat relaties tussen tabellen geldig blijven. Een cruciale vraag is: wat gebeurt er met de 'child' records (bv. de Boeken) als het 'parent' record (bv. de Series) waarnaar ze verwijzen, wordt verwijderd?.[1] Dit gedrag wordt gedefinieerd door de ON DELETE clausule van de Foreign Key.[47] De keuze tussen deze opties is een *business* beslissing, geen technische.[1]

5.4.1 Delete of target restricted (DTR) / ON DELETE RESTRICT

- **Functioneel:** De database *weigert* het 'parent' record te verwijderen zolang er nog 'child' records naar verwijzen. De gebruiker krijgt een foutmelding.[1]
- **Toepassing (Richtlijn):** Dit is de **veiligste** en **standaardoptie**.[49] Het dwingt de applicatie of gebruiker om eerst de afhankelijke records (de boeken) te verwijderen of aan een andere serie te koppelen voordat de serie zelf verwijderd kan worden. Gebruik dit voor alle kritieke data waar het 'kind' een zelfstandige waarde heeft.
- **Voorbeeld:** Een Klant met Bestellingen. Het systeem moet *weigeren* de Klant te verwijderen zolang er nog financiële Bestellingen aan gekoppeld zijn.[49]

5.4.2 Delete of target cascades (DTC) / ON DELETE CASCADE

- **Functioneel:** Het verwijderen van het 'parent' record (bv. een Order) activeert een kettingreactie die *automatisch* alle bijbehorende 'child' records (bv. alle Orderlijn records voor die order) ook verwijdert.[1]
- **Gevaar:** Deze optie is "heel gevaarlijk". Een foutieve verwijdering kan onomkeerbaar dataverlies veroorzaken ("SQL HEEFT GEEN UNDO").[1]
- **Toepassing (Richtlijn):** Mag "bijna uitsluitend in associatietabellen gebruikt worden".[1] Gebruik dit *enkel* voor data die *conceptueel eigendom* is van de 'parent' en geen zelfstandig bestaansrecht heeft (bv. zwakke entiteiten of associatietabellen).[49]
- **Voorbeeld:** Een Bestelling en zijn Bestelregels. Als de Bestelling wordt geannuleerd en verwijderd, zijn de bijbehorende Bestelregels nutteloos en *moeten* ze mee verwijderd worden.[1]

5.4.3 Delete of target nullifies (DTN) / ON DELETE SET NULL

- **Functioneel:** Het verwijderen van het 'parent' record (bv. Series) zorgt ervoor dat de FK-kolom (seriesId) in alle afhankelijke 'child' records (de Boeken) automatisch op $NULL$ wordt gezet.[1] De boeken blijven bestaan, maar zijn nu 'serie-loos'.
- **Voorwaarde (Regel):** Dit is *enkel mogelijk* indien de relatie optioneel is (Aard = 0) en de FK-kolom dus **Nulls Allowed (NA)** is. Als de FK {NNA} was, zou deze actie falen.[1]
- **Toepassing (Richtlijn):** Gebruik dit voor *optionele associaties* waarbij het 'kind' prima zelfstandig kan bestaan, maar de koppeling mag vervallen.[49]
- **Voorbeeld:** De unaire relatie Werknemer (zie 4.5.1). Als een Manager wordt verwijderd, wil men niet al zijn ondergeschikten verwijderen (geen DTC). Men wil de managerId van deze werknemers op $NULL$ zetten, waardoor ze 'manager-loos' worden totdat ze opnieuw toegewezen worden.[50]

5.5 Overige Attribuut- en Tabelconstraints (Het Integriteitsframework)

Naast PK, FK, en NULL-specificaties, zijn er drie cruciale constraints die de *Domain Integrity* (de geldigheid van een waarde) en *Business Rule Integrity* (de naleving van bedrijfsregels)

afdwingen.[42]

5.5.1 De UNIQUE Constraint

- **Definitie:** Dwingt af dat elke waarde in een kolom (of een combinatie van kolommen) uniek moet zijn binnen de tabel. Geen duplicaten toegestaan.[16]
- **Doel:** Dit is de technische implementatie van een **Alternatieve Sleutel (AK)** (zie 3.3). Het dwingt een *business rule* af, los van de PK.[16]
- **Voorbeeld:** In een Gebruiker tabel is id de PK, maar email moet ook uniek zijn. Dit wordt afgedwongen met een UNIQUE constraint.[21]
- **SQL Syntax:**

SQL

```
CREATE TABLE Gebruiker (
    id INT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,

    -- Out-of-line syntax (geeft de constraint een naam)
    CONSTRAINT AK_Gebruiker_Email UNIQUE (email)
);

-- Of In-line syntax:
-- email VARCHAR(255) NOT NULL UNIQUE
```

21

5.5.2 De CHECK Constraint

- **Definitie:** Een CHECK constraint is een krachtige regel die afdwingt dat de waarde in een kolom moet voldoen aan een specifieke Booleaanse voorwaarde.[56]
- **Doel:** Dwingt *Domain Integrity* af. Het voorkomt *semantisch* ongeldige data die het datatype wel zou toestaan (bv. een $integer$ datatype staat -5 toe, maar een CHECK (leeftijd > 0) voorkomt dit).[57]

- **Voorbeelden:**
  - prijs DECIMAL(10, 2) CHECK (prijs > 0) [56]
  - geslacht CHAR(1) CHECK (geslacht IN ('M', 'V', 'X'))
  - eindDatum DATE CHECK (eindDatum >= beginDatum) (een multi-kolom 'table constraint')
- **SQL Syntax:**

SQL

```
CREATE TABLE Product (
  id INT PRIMARY KEY,
  prijs DECIMAL(10, 2) NOT NULL,
  kostprijs DECIMAL(10, 2) NOT NULL,

  -- Dwingt af dat prijs positief moet zijn
  CONSTRAINT CK_Product_Prijs CHECK (prijs > 0),

  -- Dwingt een business rule af (marge moet positief zijn)
  CONSTRAINT CK_Product_Marge CHECK (prijs > kostprijs)
);
```

56

5.5.3 De DEFAULT Constraint

- **Definitie:** Specificeert een *standaardwaarde* die automatisch door de database wordt ingevoegd wanneer een INSERT-statement *geen* expliciete waarde voor die kolom opgeeft.[58]
- **Doel:** Vereenvoudigt data-invoer en, nog belangrijker, voorkomt ongewenste $NULL$-waarden in {NA}-kolommen (waardoor de 3VL-problematiek uit 5.3 wordt vermeden).[58]
- **Voorbeelden:**
  - isActief BIT DEFAULT 1 (Standaard is 'true')
  - status VARCHAR(20) DEFAULT 'In behandeling'
  - creatieDatum DATETIME DEFAULT GETDATE() (Gebruikt een database-functie)
- **SQL Syntax:**

SQL

```sql
CREATE TABLE Bestelling (
    id INT PRIMARY KEY,
    klantId INT NOT NULL,
    -- Als de INSERT geen orderDatum specificeert,
    -- wordt de huidige datum/tijd automatisch ingevuld
    orderDatum DATE NOT NULL DEFAULT GETDATE(),

    status VARCHAR(50) NOT NULL DEFAULT 'In behandeling'
);
```


5.6 Samenvattende Tabel: Het Data Integriteitsframework

De verschillende constraints werken samen om de database te beschermen. Ze kunnen worden gecategoriseerd op basis van het type integriteit dat ze bewaken.

Integriteitstype	**Constraint**	**Doel (Beschermt wat?)**	**SQL Voorbeeld**
Entity Integrity	PRIMARY KEY	De *identiteit* van een rij. Garandeert uniciteit.	id INT PRIMARY KEY
Entity Integrity	NOT NULL {NNA}	Het *bestaan* van een waarde. Voorkomt 'onbekend'.	naam VARCHAR(100) NOT NULL
Referential Integrity	FOREIGN KEY	De *validiteit van relaties*. Voorkomt 'wees'-records.	FOREIGN KEY (klantId) REFERENCES

			Klant(id)
Referential Integrity	ON DELETE Specs	Het *gedrag* bij het verwijderen van een 'ouder'.	... ON DELETE RESTRICT
Domain Integrity	Datatype	De *syntaxis* van een waarde.	leeftijd INT
Domain Integrity	CHECK	De *semantiek* van een waarde (business rule).	CHECK (leeftijd >= 18)
Domain Integrity	DEFAULT	De *standaardwaarde* (voorkomt $NULL$).	isActief BIT DEFAULT 1
Business Integrity	UNIQUE (AK)	De *uniciteit* van een business-attribuut.	email VARCHAR(255) UNIQUE

Deel 6: Van Theorie naar Praktijk: De SQL-Vertaling (Mapping ERD naar SQL)

Het fysieke ERD dient als directe blauwdruk voor de CREATE TABLE statements in SQL.

6.1 Conceptuele Mapping: Van ERD-diagram naar CREATE TABLE

- Elke **Entiteit** in het PDM wordt een CREATE TABLE Schemanaam.Tabelnaam statement.[1]
- Elk **Attribuut** wordt een Kolomdefinitie binnen de () met de syntaxis: veldnaam datatype.[1]
- ERD: naam: string {NNA} $\rightarrow$ SQL: naam varchar(100) NOT NULL.[1]
- ERD: nicknaam: string {NA} $\rightarrow$ SQL: nicknaam varchar(100) NULL.[1]

6.2 Vertaling van Sleutels en Attribuut-Constraints

- **Primaire Sleutel (PK):**
  - CONSTRAINT PK_Tabelnaam PRIMARY KEY (id).[1]
- **Surrogaat Sleutel (Auto-nummering):**
  - id int IDENTITY(1,1) (SQL Server syntax, begint bij 1 en telt per 1 op).[1]
- **Alternatieve Sleutel (AK):**
  - CONSTRAINT UK_Tabelnaam_Veld UNIQUE (veldnaam).[1]
- **Check Constraint (Waardebeperking):**
  - CONSTRAINT CK_Persoon_geslacht CHECK (geslacht = 'M' OR geslacht = 'V').[1]
- **Default Constraint (Standaardwaarde):**
  - CONSTRAINT DF_Persoon_land DEFAULT 'België'.[1]

6.3 Vertaling van Relaties: De FOREIGN KEY en REFERENCES Clausule

De FK-relatie uit het ERD wordt vertaald met een FOREIGN KEY constraint, die gebruik maakt van de REFERENCES clausule om de 'parent' tabel aan te duiden.[1]

- **Algemene Syntaxis:**
  - CONSTRAINT FK_HuidigeTabel_ParentTabel FOREIGN KEY (lokale_fk_kolom) REFERENCES ParentTabel (parent_pk_kolom)
- **Voorbeeld 1:**
  - CONSTRAINT FK_Muziek_Uitvoerder FOREIGN KEY (uitvoerderId) REFERENCES Les.Uitvoerder(id) [1]

6.4 Vertaling van Referentiële Integriteit: De ON DELETE Clausules

De DTC/DTR/DTN-regels uit het ERD worden als volgt toegevoegd aan het einde van de FOREIGN KEY-definitie [1]:

- **DTC $\rightarrow$ ON DELETE CASCADE**
  - SQL: ... REFERENCES Les.Order(id) ON DELETE CASCADE.[1]
- **DTR $\rightarrow$ (Geen toevoeging)**

    - SQL: Dit is het standaardgedrag van de meeste RDBMS'en; er hoeft niets te worden toegevoegd.[1] (Soms expliciet: ON DELETE RESTRICT).
- **DTN $\rightarrow$ ON DELETE SET NULL**
    - SQL: ... REFERENCES Les.Product(id) ON DELETE SET NULL.[1]

6.5 De Bouwvolgorde: CREATE en DROP

Vanwege de FOREIGN KEY constraints (en de DTR-regel) is de volgorde waarin tabellen worden aangemaakt en verwijderd cruciaal.

- **CREATE TABLE Volgorde (Top-Down):** [1]
    1. Maak eerst de 'parent' tabellen (de '1'-kant, tabellen zonder FKs), bv. Klant, Gerecht.
    2. Maak daarna de 'child' tabellen (de '*'-kant, tabellen met FKs), bv. Bestelling (die verwijst naar Klant).
    3. Maak als laatste de associatietabellen, bv. BestellingGerecht (verwijst naar Bestelling en Gerecht).
- **DROP TABLE Volgorde (Bottom-Up):** [1]
    1. De volgorde is exact omgekeerd aan de creatie.
    2. Verwijder eerst de tabellen die verwijzen (associatietabellen zoals BestellingGerecht).
    3. Verwijder daarna de 'child' tabellen (Bestelling).
    4. Verwijder als laatste de 'parent' tabellen (Klant, Gerecht).

Conclusie

ERD-modellering is een gestructureerd en progressief proces dat evolueert van een abstract conceptueel model (de 'wat'-vraag) naar een gedetailleerd fysiek model (de 'hoe'-vraag). Elke stap voegt een diepere laag van specificatie toe: van het definiëren van attributen (met aandacht voor het vermijden van samengestelde en berekende data), tot het architecturaal opzetten van sleutels (met een voorkeur voor surrogaat id sleutels), en het nauwkeurig definiëren van relaties.

De kern van een robuust model ligt in het correct toepassen van de regels voor de plaatsing van Foreign Keys (altijd aan de 'veel'-kant) en het oplossen van $M:N$-relaties via associatie-entiteiten. De uiteindelijke specificatie van datatypes, Null-toegestaanheid (NA/NNA) en On Delete-gedrag (DTC/DTR/DTN) vertaalt de businesslogica naar afdwingbare integriteitsregels. Een correcte vertaling van dit fysieke model naar SQL, met inachtneming

van de creatie- en verwijderingsvolgorde, garandeert een stabiele, efficiënte en integere database.

Geciteerd werk

1. 01_ERD_Entiteiten_en_attributen.pdf [1]

Geciteerd werk

1. ERD-Modellering_ Gedetailleerde Examenvoorbereiding.pdf
2. Conceptual model vs Logical model vs Physical model - Stack Overflow, geopend op november 6, 2025, https://stackoverflow.com/questions/25004683/conceptual-model-vs-logical-model-vs-physical-model
3. Logical vs Physical Data Model - Difference in Data Modeling - Amazon AWS, geopend op november 6, 2025, https://aws.amazon.com/compare/the-difference-between-logical-and-physical-data-model/
4. Relational model - Wikipedia, geopend op november 6, 2025, https://en.wikipedia.org/wiki/Relational_model
5. A Relational Model of Data for Large Shared Data Banks, geopend op november 6, 2025, https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf
6. Origins of Logical and Physical Data Modeling | EWSolutions, geopend op november 6, 2025, https://www.ewsolutions.com/origins-of-logical-and-physical-data-modeling/
7. Relational or Logical? - Data Model Collective, geopend op november 6, 2025, https://datamodelcollective.eu/index.php/2023/10/23/relational-or-logical/
8. Wat is datamodel? | OVHcloud Nederland, geopend op november 6, 2025, https://www.ovhcloud.com/nl/learn/what-is-data-modeling/
9. Conceptual vs Logical vs Physical Data Models: What's the Difference? - ThoughtSpot, geopend op november 6, 2025, https://www.thoughtspot.com/data-trends/data-modeling/conceptual-vs-logical-vs-physical-data-models
10. terminology - Differences between key, superkey, minimal superkey ..., geopend op november 6, 2025, https://stackoverflow.com/questions/6951052/differences-between-key-superkey-minimal-superkey-candidate-key-and-primary-k
11. What is the difference between candidate key, super key and alternate key. - Reddit, geopend op november 6, 2025, https://www.reddit.com/r/Database/comments/ezwco8/what_is_the_difference_between_candidate_key/
12. Super Key, Candidate Key and Primary Key - Data Science Duniya, geopend op

november 6, 2025, https://ashutoshtripathi.com/gate/dbms/normalization-normal-forms/super-key-candidate-key-and-primary-key/
13. Difference between Candidate, Alternate and Secondary Keys in Databases - Reddit, geopend op november 6, 2025, https://www.reddit.com/r/mysql/comments/82z2c2/difference_between_candidate_alternate_and/
14. Super Key vs Candidate Key vs Primary Key. Understand the difference. - YouTube, geopend op november 6, 2025, https://www.youtube.com/watch?v=_UqSgxC1kK8
15. What are the main differences between a primary key and a unique constraint?, geopend op november 6, 2025, https://stackoverflow.com/questions/37388100/what-are-the-main-differences-between-a-primary-key-and-a-unique-constraint
16. Unique constraints and check constraints - SQL Server - Microsoft Learn, geopend op november 6, 2025, https://learn.microsoft.com/en-us/sql/relational-databases/tables/unique-constraints-and-check-constraints?view=sql-server-ver17
17. Difference between Primary key and Unique key - GeeksforGeeks, geopend op november 6, 2025, https://www.geeksforgeeks.org/dbms/difference-between-primary-key-and-unique-key/
18. A beginner question on Primary Key VS Unique Key, Not Null.. : r/SQL - Reddit, geopend op november 6, 2025, https://www.reddit.com/r/SQL/comments/qs4enx/a_beginner_question_on_primary_key_vs_unique_key/
19. Alternate Key Explained: Types, Differences, & Examples - upGrad, geopend op november 6, 2025, https://www.upgrad.com/blog/what-is-an-alternate-key/
20. Different keys in SQL, Primary Key, Candidate Key, Foreign Key - Analytics Vidhya, geopend op november 6, 2025, https://www.analyticsvidhya.com/blog/2020/07/difference-between-sql-keys-primary-key-super-key-candidate-key-foreign-key/
21. SQL UNIQUE - Syntax, Use Cases, and Examples - Hightouch, geopend op november 6, 2025, https://hightouch.com/sql-dictionary/sql-unique
22. SQL UNIQUE Constraint (With Examples) - Programiz, geopend op november 6, 2025, https://www.programiz.com/sql/unique
23. SQL Composite Key - Programiz, geopend op november 6, 2025, https://www.programiz.com/sql/composite-key
24. Defining Composite Primary and Foreign Keys - IBM, geopend op november 6, 2025, https://www.ibm.com/docs/en/informix-servers/14.10.0?topic=format-defining-composite-primary-foreign-keys
25. One-to-One Database Relationships: Complete Implementation Guide - Beekeeper Studio, geopend op november 6, 2025, https://www.beekeeperstudio.io/blog/one-to-one-database-relationships-compl

ete-guide
26. Relaties tussen tabellen in een Access-database definiëren - Microsoft 365 Apps, geopend op november 6, 2025, https://learn.microsoft.com/nl-nl/office/troubleshoot/access/define-table-relationships
27. Eén-op-veel-relaties | Claris FileMaker Pro Help, geopend op november 6, 2025, https://help.claris.com/nl/pro-help/content/one-to-many-relationships.html
28. Veel-op-veel-relaties | Claris FileMaker Pro Help - Claris Help Center, geopend op november 6, 2025, https://help.claris.com/nl/pro-help/content/many-to-many-relationships.html
29. Waarom geen 1-op-veel relaties behandelen als veel-op-veel? : r/learnprogramming, geopend op november 6, 2025, https://www.reddit.com/r/learnprogramming/comments/axv7tf/why_not_treat_1many_relations_as_manymany/?tl=nl
30. Understanding Self Join and Self-Referencing (Recursive) Relationships in Databases and use it with Hibernate | by Ahad Ali | Medium, geopend op november 6, 2025, https://medium.com/@linkonahad10/understanding-self-join-and-self-referencing-recursive-relationships-in-databases-and-use-it-with-de0d28fbd3a4
31. What is an industry standard name for this self-referencing foreign key relationship?, geopend op november 6, 2025, https://dba.stackexchange.com/questions/341565/what-is-an-industry-standard-name-for-this-self-referencing-foreign-key-relation
32. SQL Recursive CTE Explained | Build Employee-Manager Hierarchy with Step-by-Step Execution! - YouTube, geopend op november 6, 2025, https://www.youtube.com/watch?v=TCBnzs2DM8w
33. How to Build an Employee-Manager Hierarchy Using Recursive CTE in SQL - Medium, geopend op november 6, 2025, https://medium.com/analytics-vidhya/how-to-build-an-employee-manager-hierarchy-using-recursive-cte-in-sql-0557dc0a8293
34. Master Employee Hierarchy in SQL: Recursive CTE Explained with Examples - YouTube, geopend op november 6, 2025, https://www.youtube.com/watch?v=Pkksi2szle0
35. SQL Tip: Self-Referential Tables - RedBit Development, geopend op november 6, 2025, https://www.redbitdev.com/post/sql-tip-self-referential-tables
36. Ternary Relationship ER Diagram - Creately, geopend op november 6, 2025, https://creately.com/diagram/example/nCDfQmT5fzU/ternary-relationship-er-diagram
37. Modelleren van ternaire relaties voor een RDBMS : r/SQL - Reddit, geopend op november 6, 2025, https://www.reddit.com/r/SQL/comments/18zhflz/modeling_ternary_relationships_for_an_rdbms/?tl=nl
38. Identificerende vs. niet-identificerende relaties: Waarom hebben ze ..., geopend op november 6, 2025, https://www.reddit.com/r/Database/comments/am5h4b/identifying_vs_non_identi

fying_relationships_why/?tl=nl
39. Zwakke entiteit in ER-diagrammen: Een voltooide gids - ClickUp, geopend op november 6, 2025, https://clickup.com/nl/blog/264674/zwakke-entiteit
40. Relationeel model - Edutorial, geopend op november 6, 2025, https://edutorial.nl/dbo/relationeel-model/
41. Wat is een relationele database - Oracle, geopend op november 6, 2025, https://www.oracle.com/nl/database/what-is-a-relational-database/
42. constraint - Oracle Help Center, geopend op november 6, 2025, https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/constraint.html
43. Database Design Follies: NULL vs. NOT NULL – SQLServerCentral, geopend op november 6, 2025, https://www.sqlservercentral.com/articles/database-design-follies-null-vs-not-null
44. When to use Null vs. N/A in columns? - Stack Overflow, geopend op november 6, 2025, https://stackoverflow.com/questions/28854903/when-to-use-null-vs-n-a-in-columns
45. Why shouldn't we allow NULLs? - Database Administrators Stack Exchange, geopend op november 6, 2025, https://dba.stackexchange.com/questions/5222/why-shouldnt-we-allow-nulls
46. NULL vs NOT NULL - sql - Stack Overflow, geopend op november 6, 2025, https://stackoverflow.com/questions/10933639/null-vs-not-null
47. mysql - Setting up table relations what do "Cascade", "Set Null" and ..., geopend op november 6, 2025, https://stackoverflow.com/questions/5383612/setting-up-table-relations-what-do-cascade-set-null-and-restrict-do
48. Difference between CASCADE and RESTRICT? SQL DDL database - Stack Overflow, geopend op november 6, 2025, https://stackoverflow.com/questions/20619211/difference-between-cascade-and-restrict-sql-ddl-database
49. Restrict vs Cascade vs Set Null in SQL: Choosing the Right Foreign Key Rule - Medium, geopend op november 6, 2025, https://medium.com/@sunnywilson.veshapogu/restrict-vs-cascade-vs-set-null-in-sql-choosing-the-right-foreign-key-rule-6d7c98484710
50. ON DELETE CASCADE vs. SET NULL - SQL - Reddit, geopend op november 6, 2025, https://www.reddit.com/r/SQL/comments/11o0fjd/on_delete_cascade_vs_set_null/
51. Difference Between ON DELETE CASCADE and ON DELETE SET NULL in DBMS, geopend op november 6, 2025, https://www.geeksforgeeks.org/dbms/difference-between-on-delete-cascade-and-on-delete-set-null-in-dbms/
52. List all table constraints (PK, UK, FK, Check & Default) in SQL Server database - Dataedo, geopend op november 6, 2025, https://dataedo.com/kb/query/sql-server/list-all-table-constraints

53. Understanding the SQL UNIQUE Constraint - DbVisualizer, geopend op november 6, 2025, https://www.dbvis.com/thetable/understanding-the-sql-unique-constraint/
54. UNIQUE Constraint - SQL - GeeksforGeeks, geopend op november 6, 2025, https://www.geeksforgeeks.org/sql/sql-unique-constraint/
55. Create unique constraints - SQL Server - Microsoft Learn, geopend op november 6, 2025, https://learn.microsoft.com/en-us/sql/relational-databases/tables/create-unique-constraints?view=sql-server-ver17
56. SQL CHECK Constraint: Definitive Guide With Examples, geopend op november 6, 2025, https://www.dbvis.com/thetable/sql-check-constraint-definitive-guide-with-examples/
57. SQL | CHECK Constraint - GeeksforGeeks, geopend op november 6, 2025, https://www.geeksforgeeks.org/sql/sql-check-constraint/
58. SQL DEFAULT - Syntax, Use Cases, and Examples | Hightouch, geopend op november 6, 2025, https://hightouch.com/sql-dictionary/sql-default
59. SQL DEFAULT Constraint (With Examples) - Programiz, geopend op november 6, 2025, https://www.programiz.com/sql/default
60. SQL | DEFAULT Constraint - GeeksforGeeks, geopend op november 6, 2025, https://www.geeksforgeeks.org/sql/sql-default-constraint/
61. Specify default values for columns - SQL Server - Microsoft Learn, geopend op november 6, 2025, https://learn.microsoft.com/en-us/sql/relational-databases/tables/specify-default-values-for-columns?view=sql-server-ver17