Inleiding databases

Inleiding databases

Van ontwerp tot praktijk

Ben Groenendijk

Boom

Voorwoord

Databases zijn niet meer weg te denken uit het dagelijks leven. Van een afgestudeerde op hbo- of wo-niveau mag worden verwacht dat deze kennis van en inzicht in databases heeft. Naast de theoretische kennis dienen ook praktische vaardigheden aanwezig te zijn.

Allereerst wordt toegelicht waarom databases worden toegepast en wat het verschil is tussen een database en een spreadsheet (Excel). Op welke manier liggen gegevens opgeslagen in een database en waarom? Ook wordt toegelicht wat 'big data' is en hoe dat werkt.

Vervolgens komt het ontwerpen van databases ter sprake. Hierbij wordt de techniek van het normaliseren in eenvoudige stappen met instructiefilmpjes toegelicht. Het ontwerp van de gegevensstructuur van de database wordt vervolgens grafisch weergegeven door middel van een entiteitrelatiediagram.

Het doorgronden hoe een database werkt lukt alleen door praktisch aan de slag te gaan. Leren autorijden vanuit een theorieboek zal niet lukken. Toen ik zelf ging leren autorijden kende ik de theorie, alle verkeersborden, alle verkeersregels, iedere knop in de auto (van mijn vader) kon ik blindelings bedienen en de wegen in de buurt kende ik uit mijn hoofd. Maar bij de eerste rijles kwam ik de straat niet uit. Aan de slag gaan met databases is dus vereist voor een goed begrip. Daarbij is het niet noodzakelijk om direct met een grote database aan de slag te gaan. Als je met vier, vijf tabellen in een database kunt werken, kun je ook met vijfhonderd tabellen werken. Praktische training vindt plaats met Microsoft Access en/of een SQL-omgeving. Het is niet de opzet om Access te leren, maar om de theorie over databases toe te passen in een database. Hierdoor krijg je meer kennis en inzicht in databases. Voor beide omgevingen zijn bestaande databases te downloaden via de bij dit boek horende website www.inleidingdatabases.nl.

In de databaseomgeving gaat de meeste aandacht uit naar het belangrijkste onderdeel: het opvragen van informatie uit de database (query's). Dat gaat van eenvoudige query's tot en met het produceren van managementinformatie. Alle facetten komen aan bod. In Access worden ook formulieren en rapporten gemaakt om professionele invoer en uitvoer te maken.

In het boek wordt ieder onderwerp in kleine stapjes met behulp van voorbeelden besproken en toegelicht, waarbij volop gebruik wordt gemaakt van relevante schermafdrukken en instructiefilmpjes (). Vervolgens worden bij ieder onderwerp opgaven aangeboden waarin de theoretische kennis direct praktisch kan worden toegepast. De uitwerkingen van de opgaven zijn ook beschikbaar via de website.

Voor het SQL-gedeelte van het boek is het niet noodzakelijk om zelf over een SQL-tool te beschikken. Op de website van dit boek worden diverse opties gegeven voor gratis te downloaden SQL-tools. Ook de handleidingen van die tools zijn daar te vinden. Voor

het Access-gedeelte van het boek dien je te beschikken over Access 2010 of hoger. De schermafdrukken zijn van Access 2016. Het maakt niet uit of je de beschikking hebt over de Engelstalige of de Nederlandstalige versie. In het boek is uitgegaan van de Nederlandstalige versie, maar steeds is ook het Engelstalige equivalent erbij gegeven.

Voor docenten is extra materiaal beschikbaar op de website, zoals presentaties in PowerPoint en diverse cases.

Vragen, opmerkingen of bedrijfstrainingen naar aanleiding van dit boek zijn welkom. Stuur deze aan: b.j.groenendijk@hr.nl

Ben Groenendijk
Februari 2017

Inhoud

Hoofdstuk 1

Databases

In dit hoofdstuk wordt duidelijk gemaakt wat een database is. Uitgelegd wordt hoe gegevens in een database worden opgeslagen en hoe die gegevens snel gevonden kunnen worden uit miljoenen regels aan gegevens. Ook het begrip 'big data' wordt toegelicht.

1.1 Gegevens

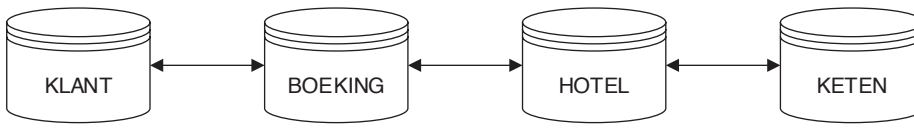
We worden dagelijks overspoeld door een enorme hoeveelheid gegevens en informatie die ons continu wordt aangeboden via internet, e-mail, Facebook, Twitter, Instagram, WhatsApp, YouTube, radio en televisie. Niet alleen in ons privéleven worden we overspoeld, ook in onze contacten met de 'zakelijke wereld' worden we er volop mee geconfronteerd. Een groot deel van de aangeboden gegevens/informatie ervaren we als niet-relevant. Maar er zijn ook zaken die we willen vastleggen. Soms omdat we daartoe verplicht zijn (bedrijven/instellingen/organisaties), soms omdat we voor ons zelf zekerheid willen (afspraken in agenda, e-mails, contactgegevens) en soms omdat we het gewoon leuk vinden (reisverslag van de vakantie, kasboek met inkomsten en uitgaven, enzovoort). Voor het vastleggen van gegevens wordt tegenwoordig vrijwel altijd gebruikgemaakt van een computer, een tablet of een smartphone; de hardware. Alleen hardware volstaat niet, er dient ook software aanwezig te zijn waarmee de gegevens kunnen worden ingevoerd en opgeslagen. Documenten kunnen we bijvoorbeeld maken en opslaan in Microsoft Word, een rekenmodel kunnen we bijvoorbeeld maken en opslaan in Microsoft Excel. Maar waar slaan Facebook, Twitter, WhatsApp, enzovoort hun gegevens op? Al die gegevens worden opgeslagen in databases. Maar ook de bestellingen bij alle webwinkels (Bol.com, Coolblue.nl, Lucardi.nl, Booking.com) worden opgeslagen in een database. Alle bedrijven/instellingen/organisaties registeren hun inkoopgegevens, productiegegevens, verkoopgegevens, financiële gegevens, patiëntgegevens, studentgegevens, gegevens over verkeersstromen, enzovoort in een database.

Aan de hand van een voorbeeld zal duidelijk gemaakt worden waarom gegevens in een database worden opgeslagen. Stel, we gaan een lang weekend weg en reserveren bij Booking.com een hotel. Op de site van Booking.com wordt incheckdatum, uitcheckdatum, land, stad en een geschikt hotel geselecteerd. Vervolgens een kamertype, aantal kamers, aantal personen per kamer, de persoonsgegevens, e-mailadres en uiteindelijk de betaalgegevens. Na een geslaagde boeking krijg je direct een mail met de boekingsgegevens in pdf-formaat. Je zou misschien denken dat Booking.com iedere boeking in hun boekingssysteem als pdf-bestand opslaat. De bestandnaam zou dan het unieke boekingsnummer kunnen zijn, zodat het ook snel teruggevonden kan worden. Het grootste nadeel van die methode is dat er geen managementinformatie of marketinginformatie uit te halen is. Bijvoorbeeld: wat is de omzet over de afgelopen maand? Wat is per land het aantal boekingen over het afgelopen jaar? Welke hotels hebben het meeste opgebracht in het laatste kwartaal? Op basis van het boekingsgedrag de klant verleiden

met een gerichte e-mail. Het zou nog kunnen bij tien boekingen per maand door de pdf-bestanden een voor een na te pluizen, maar bij duizenden boekingen per dag is dat onuitvoerbaar! Met een database kan dat wel.

1.2 Gegevens in een database

In een database worden gegevens (data) in logische groepen opgeslagen. Een vereenvoudigd databasemodel van ons voorbeeld is weergegeven in figuur 1.1.



Figuur 1.1

Er is een bestand met alleen (geregistreerde) klantgegevens, groep *Klant*. Verder een bestand met alleen boekingsgegevens, groep *Boeking*, een bestand met gegevens van de te boeken hotels, groep *Hotel* en als laatste een bestand met de gegevens van alle hotelketens (eigenaren van de hotels), groep *Keten*. Een hotel heeft een eigenaar. Een eigenaar kan een of meerdere hotels bezitten. Zo heeft bijvoorbeeld eigenaar Hilton of Ibis vele hotels, vandaar de naam hotelketen.
In de groep *Klant* worden de basisgegevens van alle klanten gestructureerd opgeslagen in de vorm van een tabel, zie figuur 1.2.

KLANT

Klantnr	Naam	Adres	Postcode	Plaats	E-mail
100001	T. de Vries	Steenstraat 41	1380 VB	Weesp	vries34@gmail.com
100002	D. Aanraad	Leiweg 12	1621 AP	Hoorn	d.aanraad@outlook.com
100003	M. den Hoed	Poststraat 111	3011 VD	Rotterdam	m.den.hoed@bijenkorf.nl
.....					

Figuur 1.2

In de kolomkoppen staan de gewenste gegevens (kenmerken, eigenschappen, elementen) van een klant. In dit voorbeeld *Klantnr*, *Naam* t/m *E-mail*. In een rij staan de klantgegevens van één klant. De gegevensgroep en de kenmerken vertellen iets over de gehele groep, alle klanten.
In de groep *Boeking* worden de basisgegevens (kenmerken) van een boeking gestructureerd opgeslagen in de vorm van een tabel, zie figuur 1.3.

BOEKING

Boekingnr	Hotelnr	Checkin	Checkout	Kamertype	Aant. pers.	Aant. kamers	Klantnr
931321100	678891	8-7-2017	10-7-2017	Suite King	2	1	100003
931321101	187612	31-12-2017	8-1-2018	DZ standaard	2	1	564162
931321102	476324	17-9-2017	19-9-2017	Luxe Twin	1	2	761321
.....							

Figuur 1.3

Bij iedere boeking worden de kenmerken *Boekingnr*, *Hotelnr* t/m *Klantnr* opgeslagen. Voor de overzichtelijkheid zijn niet alle kenmerken getoond, zoals boekingsdatum, betaaldatum, betaalwijze, enzovoort. Van het geboekte hotel wordt alleen het hotelnummer opgenomen. Van de klant die geboekt heeft wordt alleen het klantnummer opgenomen.
In de groep *Hotel* worden de basisgegevens (kenmerken) van alle hotels gestructureerd opgeslagen in de vorm van een tabel, zie figuur 1.4.

HOTEL

Hotelnr	Naam	Adres	Postcode	Plaats	Land	Keten-nr
187612	Gletscherblick	Poststrasse 8	6561	Ischgl	Oostenrijk	12349
187613	Rosewood	Verdun 1	7690	Franschhoek	Zuid-Afrika	42176
187614	Ibis Schiphol	Schipholweg 181	1171 PK	Badhoevedorp	Nederland	31254
.....						

Figuur 1.4

Naast de basishotelkenmerken is ook het ketennummer opgenomen, zodat bekend is bij welke hotelketen het hotel hoort.
In groep *Keten* worden de basisgegevens (kenmerken) van alle hotelketens (eigenaren) gestructureerd opgeslagen in de vorm van een tabel, zie figuur 1.5.

KETEN

Ketennr	Naam	Adres	Postcode	Plaats	Land	Provisie
31254	Ibis SA	Alpine 12	04281	Parijs	Frankrijk	12%
31255	Hilton	Beverly Hills 18	90210	Los Angeles	V.S.	15%
31256	NH Hotels	Calle gran vía 11	28013	Madrid	Spanje	12,5%
.....						

Figuur 1.5

Booking.com krijgt per boeking een vergoeding (provisie). De facturen worden gestuurd aan de eigenaar van het hotel.

De gegevens in de gegevensgroepen zijn zo efficiënt mogelijk opgeslagen. Zo wordt bij een boeking in de gegevensgroep *Boeking* alleen het hotelnummer opgeslagen en niet alle hotelgegevens. Zou je wel alle hotelgegevens iedere keer opslaan in gegevensgroep *Boeking*,dan zouden dezelfde hotelgegevens meerdere keren voorkomen in gegevensgroep *Boeking*. Als het hotel tien keer wordt geboekt, zouden de hotelgegevens tien keer voorkomen in *Boeking*. Dezelfde gegevens worden dan meerdere keren opgeslagen. Dit wordt *redundantie* genoemd. Met redundantie wordt bedoeld dat dezelfde gegevens meerdere keren liggen opgeslagen. Het is een bekend begrip bij databases; redundantie moet voorkomen worden. Als je dezelfde gegevens meerdere keren gaat opslaan, bijvoorbeeld klantgegevens, en de klant verhuist of krijgt een ander telefoonnummer, moet je die wijzigingen ook op meerdere plaatsen veranderen. Als je dat niet doet, ontstaat een ernstig probleem dat *inconsistentie* wordt genoemd. Inconsistentie betekent tegenstrijdigheid. Er zijn dan bijvoorbeeld verschillende adressen van een klant bekend, de database wordt dan onbetrouwbaar. Je denkt misschien dat zoiets niet voorkomt, maar het komt in organisaties toch veel voor. Bij fusies van organisaties heb je al twee databases en die moeten wel op een juiste manier samengevoegd worden. Vaak lukt dat niet goed. Zelf zie ik bij een niet nader te noemen bank op internet, bij persoonlijke gegevens, een adres staan waar ik al tien jaar niet meer woon (adreswijziging doorgegeven). Post van diezelfde bank krijg ik wel op het goede adres. De adresgegevens zijn dus minimaal op twee plaatsen opgeslagen. Gegevens efficiënt opslaan zonder redundantie is dus belangrijk binnen databases.

In ons vereenvoudigde datamodel zijn voor de overzichtelijkheid maar een aantal kolomkoppen per gegevensgroep opgenomen. In werkelijkheid zijn er tientallen kolomkoppen (kenmerken) van een gegevensgroep. Ook het aantal gegevensgroepen is in het voorbeeld voor de overzichtelijkheid klein gehouden. Het aantal tabellen in professionele database loopt snel op. Bijvoorbeeld het *ERP-pakket* Dynamics NAV van Microsoft voor het midden- en kleinbedrijf. Met die software kun je de volledige bedrijfsvoering verwerken: inkoop, verkoop, productie, voorraadbeheer, financiële administratie, personeelszaken (hr – human resources), enzovoort. Dat pakket heeft zo'n duizend tabellen en de tabel *Klant* bezit al meer dan vijftig kenmerken. Zo'n softwarepakket voor het grootbedrijf, bijvoorbeeld SAP, heeft al meer dan tweeduizend tabellen. Het principe blijft overigens precies hetzelfde. Om informatie uit de database te halen heb je nooit alle tabellen nodig, vaak maar een paar.

Kijk op www.inleidingdatabases.nl voor een demonstratie van zo'n professionele database: *professionele database*.

1.2.1 Relaties tussen gegevensgroepen

De gegevens van alle boekingen in ons (vereenvoudigde) voorbeeld liggen opgeslagen in vier gegevensgroepen (tabellen) in de database. Tussen die gegevensgroepen in de database zijn relaties (koppelingen). De groep *Klant* en de groep *Boeking* zijn gekoppeld via het kenmerk *Klantnr*, zie figuur 1.2 en 1.3.

Selecteer je een willekeurige rij in de tabel *Boeking*, dan wordt automatisch op basis van het klantnummer in de rij van de tabel *Boeking* de bijbehorende klant geselecteerd in de tabel *Klant*, zie figuur 1.6.

BOEKING

Boekingnr	Hotelnr	Checkin	Checkout	Kamertype	Aant. pers.	Aant. kamers	Klantnr
931321100	678891	8-7-2017	10-7-2017	Suite King	2	1	100003

KLANT

Klantnr	Naam	Adres	Postcode	Plaats	E-mail
100003	M. den Hoed	Poststraat 111	3011 VD	Rotterdam	m.den.hoed@bijenkorf.nl

Figuur 1.6

Maar het werkt ook andersom. Selecteer je een willekeurige rij in de tabel *Klant*, dan wordt automatisch op basis van het klantnummer in de rij van de tabel *Klant* de bijbehorende boeking(en) geselecteerd in de tabel *Boeking*, zie figuur 1.7.

KLANT

Klantnr	Naam	Adres	Postcode	Plaats	E-mail
100003	M. den Hoed	Poststraat 111	3011 VD	Rotterdam	m.den.hoed@bijenkorf.nl

BOEKING

Boekingnr	Hotelnr	Checkin	Checkout	Kamertype	Aant. pers.	Aant. kamers	Klantnr
927632251	515453	3-2-2015	17-2-2015	Royal Twin	2	1	100003
931321100	678891	8-7-2017	10-7-2017	Suite King	2	1	100003
931334241	231765	8-12-2017	2-1-1018	Exel Prime	2	1	100003

Figuur 1.7

De werking van de relaties tussen de tabellen in een database is denkbeeldig voor te stellen door de tabellen te vertalen naar Microsoft Excel. In het eerste tabblad in Excel maak je de tabel *Klant*, in het tweede tabblad de tabel *Boeking*, in het derde tabblad de tabel *Hotel* en ten slotte een tabblad *Keten* met de hotelketengegevens. Alle tabbladen zijn gevuld met tienduizenden regels. Als Excel hetzelfde zou werken als een database en ik zou een willekeurige klant selecteren in het tabblad *Klant* en vervolgens selecteer ik het tabblad *Boeking*, dan zie je alleen nog maar de boekingen van de geselecteerde klant uit het tabblad *Klant*. De tienduizenden boekingsrijen zijn dan automatisch gefilterd op het geselecteerde klantnummer uit de tabel *Klant*. Stel, er zijn drie boekingen van de geselecteerde klant zichtbaar in het tabblad *Boeking*. Selecteer een van de drie boekingen en selecteer vervolgens het tabblad *Hotel*. Je ziet nu nog maar één regel, namelijk het

hotel dat geboekt is. De koppeling (relatie) tussen de tabel *Boeking* en de tabel *Hotel* is via het kenmerk *Hotelnr*. Er wordt in het tabblad *Hotel* van Excel dan automatisch gefilterd op het geselecteerde hotelnummer uit het tabblad *Boeking*. Hetzelfde geldt voor het tabblad *Keten*. De koppeling tussen het tabblad *Hotel* en het tabblad *Keten* is via het kenmerk *Ketennr*. Zou je vervolgens het tabblad *Keten* selecteren, dan zie je ook maar één regel.
De relaties tussen de tabellen werken automatisch alle kanten op. Zou je bijvoorbeeld een willekeurige hotelketen selecteren in het tabblad *Keten* en je selecteert vervolgens het tabblad *Hotel*, dan zie je alleen nog maar de hotels van de geselecteerde keten uit het tabblad *Keten*. Selecteer je vervolgens een willekeurig hotel uit de keten in het tabblad *Hotel* en je gaat vervolgens naar het tabblad *Boeking*, dan zie je alle boekingen die gemaakt zijn naar het hotel dat op het tabblad *Hotel* is geselecteerd. Selecteer je daarna een willekeurige boeking en je kijkt vervolgens op het tabblad *Klant*, dan zie je maar één klant. Het is de klant van de geselecteerde boeking uit het tabblad *Boeking*.
Door de gegevensgroepen voor te stellen als tabellen in Excel is de werking van een database goed voor te stellen. Uiteraard kan dat niet gerealiseerd worden in Excel en bovendien is Excel beperkt. Het maximumaantal rijen in Excel is 1.048.576. Voor de tabellen *Klant* en *Keten* is dat voldoende. Maar Booking.com heeft wereldwijd al meer kan een miljoen hotels die geboekt kunnen worden, dus dat gaat al krap worden en het aantal boekingen wereldwijd gaat ver voorbij het miljoen. Het aantal boekingen wereldwijd en dus rijen in de gegevensgroep *Boeking* is 650.000 per dag! Een database heeft die beperkingen niet. De enige beperkingen zijn de beschikbare hoeveelheid geheugen en systeembronnen. Nog een belangrijk verschil tussen Excel en een database is dat alle rijen in een tabel uniek zijn! In Excel zou je twee precies dezelfde rijen kunnen invoeren in een tabel. Maar in een database kan dat niet, iedere rij in de tabel *moet* uniek zijn. Wordt een unieke regel ingevoerd, dan ontstaat automatisch een foutmelding in een database. Door het toekennen van unieke klantnummers, boekingsnummers, hotelnummers, enzovoort worden de rijen in een database uniek gemaakt.

Kijk op www.inleidingdatabases.nl voor een filmpje over relaties tussen gegevensgroepen in een database: *relaties tussen gegevensgroepen*.

1.2.2 Definitie database

We kunnen nu een definitie geven van een database: een verzameling bij elkaar behorende gegevensgroepen inclusief hun onderlinge relaties.

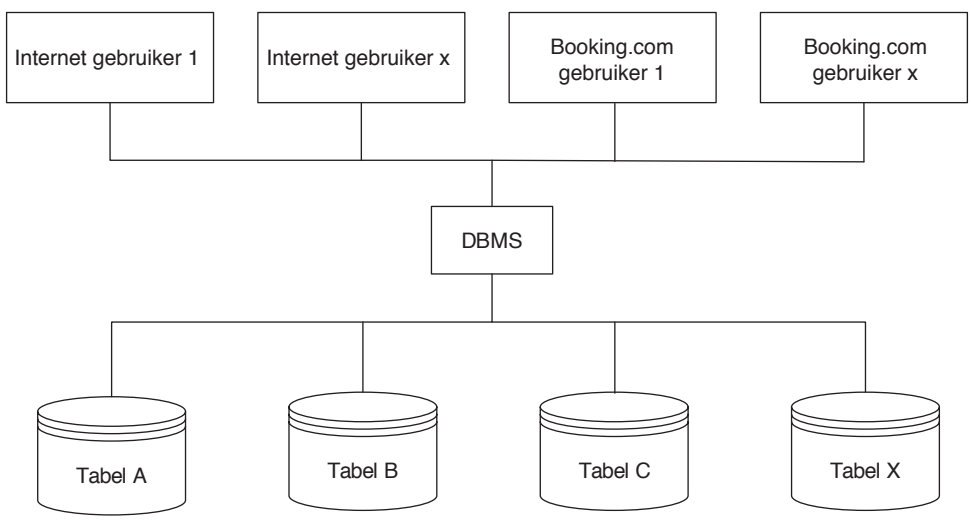
Ook belangrijk om in gedachten te houden is dat in databases gegevens, de tabellen, en de in- en uitvoer van die gegevens gescheiden zijn. De gegevens liggen in een relationele database gestructureerd opgeslagen. Maar zo zijn ze niet te tonen aan gebruikers van die gegevens. Dan zou je alleen maar tabellen zien en dat is niet echt gebruiksvriendelijk. Op internet, in apps of bij standaardsoftware (applicaties) die gebruikmaken van databases krijg je altijd een mooi opgemaakt scherm te zien, waarin je de gegevens kunt invoeren en/of selecteren. De gegevens op het scherm komen vaak uit meerdere tabellen uit de database. Op het moment dat je inlogt zijn bijvoorbeeld je adresgegevens bekend. Bij Booking.com komen de hotels uit de tabel *Hotel*. Als je boekt gaan verschillende gegevens naar de tabel *Boeking*. Dat geldt eigenlijk voor iedere app(licatie) die gebruiktmaakt van een database. Meestal na het selecteren van OK worden de gegevens verwerkt.

Achter de schermen worden de gegevens dan automatisch in de juiste tabellen gezet. Het scherm waarop je gegevens invoert wordt binnen databases een *Formulier* genoemd. Hetzelfde geldt voor het presenteren van de gegevens, bijvoorbeeld week- of maandrapportages, een factuur of een bevestiging van een boeking. Die krijg je niet gepresenteerd als een tabel uit de database. Week- of maandrapportages wil je bijvoorbeeld tonen met een grafiek of dashboard (alle belangrijke informatie in één scherm). Een factuur moet eruitzien als een factuur met aanhef, adresgegevens, bestelde producten, de prijs, btw, enzovoort. Die gegevens komen ook uit meerdere tabellen en worden dan op de gewenste manier getoond. Het tonen van de gegevens wordt binnen databases een *Rapport* genoemd. De gegevens liggen gestructureerd opgeslagen in tabellen, maar het invoeren van de gegevens of het tonen van die gegevens kan op ieder gewenste manier.

Kijk op www.inleidingdatabases.nl voor een demonstratie van de gegevens, het invoeren van die gegevens in een formulier en het tonen van de gegevens in een rapport: *invoer en uitvoer database*.

1.3 Werking database

Historisch gezien zijn er verschillende typen databases. Eerst kwam de hiërarchische database, toen de netwerkdatabase en vervolgens de relationele database. De beschrijving van het vorige hoofdstuk betreft de relationele database. De relationele database wordt veruit het meest toegepast. Bij een relationele database hoort een *databasemanagementsysteem*, afgekort *DBMS*. Dit systeem vormt een sluis tussen de gebruikers van de database en de gegevensbestanden, zie figuur 1.8.



Figuur 1.8

Als we het voorbeeld van Booking.com nog even aanhouden, dan zijn er internetgebruikers die via de webpagina hotelinformatie opvragen uit de database en internetgebrui-

kers die een boeking plaatsen. Maar bij het bedrijf Booking.com maakt men natuurlijk ook gebruik van de database door financiële overzichten op te stellen, facturen te maken en die sturen naar de hotelketens of managementrapportages op te stellen. In ons vereenvoudigde voorbeeld hebben we maar vier gegevensgroepen gebruikt, maar in werkelijkheid zijn er tientallen gegevensgroepen. Die gegevensstructuren zijn eerst ontworpen 'op papier', dat wordt het *conceptueel* of *functioneel datamodel* genoemd. Het ontwerpen van de gegevensstructuren wordt in hoofdstuk 2 toegelicht. Het conceptuele datamodel wordt vervolgens via het DBMS van de database gebouwd in de database. Dat wordt het volledige datamodel van de database genoemd. Het DBMS bewaakt ook wie welke gegevens mag inzien. Zo kan een internetgebruiker wel de basisgegevens uit de tabel *Hotel* zien, maar niet bijvoorbeeld de contactpersoon van dat hotel voor Booking.com of de vergoeding die Booking.com ontvangt over een boeking. Maar ook niet alle medewerkers van Booking.com mogen alle gegevens zien. Een medewerker van de klantenservice ziet bijvoorbeeld wel wat geboekt en betaald is, maar niet het creditcardnummer. Zo kan bijvoorbeeld de personeelsfunctionaris het salaris van een medewerker zien, maar andere werknemers die ook personeelsgegevens gebruiken niet. Via *autorisatie* (*permissies*) kan aangegeven worden wat een gebruiker wel en niet mag. Bijvoorbeeld hotelgegevens wel lezen, maar niet bewerken. De tabellen en kenmerken (kolommen) binnen een tabel van de database die een gebruiker kan 'zien' wordt *view* op de database genoemd. Diegene de database onderhoud wordt *databaseadministrator* genoemd.

1.4 Gegevens zoeken in een database

Het DBMS zoekt de gewenste gegevens in de gestructureerde tabellen. Dat gaat razendsnel. Als je in de tabel *Klant* een willekeurige klant selecteert, dan is in tabel *Boeking* ook automatisch via het *Klantnr* uit de tabel *Klant* de boekingen zichtbaar van die klant, zie figuur 1.7. Hoe kunnen die gegevens zo snel gevonden worden? Stel, we hebben klantnummer 100003 geselecteerd in de tabel *Klant*. In de kolom *Klantnr* van de tabel *Boeking* moeten alle rijen gevonden worden van klantnummer 100003. Laten we het eenvoudig houden en uitgaan van een miljoen rijen in tabel *Boeking*. In werkelijkheid 650.000 rijen per dag erbij, dus na anderhalve dag is dat al bereikt. De computer moet nu een miljoen rijen doorlopen om alle boekingen van klant 100003 te achterhalen. We maken een rekensom om een schatting te maken hoe lang dat zou duren. De snelheid van een harde schijf is ongeveer 10 milliseconde per rij uit de tabel (een solid state disk (SSD) is een factor 5 tot 10 sneller). Dat is 1/100 van een seconde, maar een miljoen keer 1/100 van een seconde is 10.000 seconden. Dat is ongeveer 2 uur en 45 minuten! Op brute kracht door veel gegevens (data) in een database heengaan werkt niet. Hoe werkt het dan? Als de gegevens in de tabel *Boeking* gesorteerd zouden liggen op klantnummer, kan sneller gezocht worden. Maar het kost veel tijd om zo'n grote tabel te sorteren. Bedenk dat het aantal kolommen in ons voorbeeld klein gehouden is, in werkelijk zijn het tientallen kolommen. Aangezien je de tabel ook in de oorspronkelijke volgorde nodig hebt, heb je dan twee keer dezelfde gegevens opgeslagen, alleen in een andere volgorde. Daar is een oplossing voor bedacht. Een database maakt automatisch 'achter de schermen' kleine *indexbestanden* aan met maar twee kolommen. In het indexbestand van de tabel *Boeking* zijn de klantnummers oplopend gesorteerd en achter het klantnummer staat het

rijnummer uit de bijbehorende tabel *Boeking*, zie figuur 1.9. Rijnummers zijn de logische nummers van de rijen, de eerste rij heeft rijnummer 1, de tweede rijnummer 2 en de laatste in ons voorbeeld van een miljoen rijen heeft rijnummer 1.000.000.

Klantnr	Rijnr
100	654321
100	765431
101	2342
.....	.....
100003	12651
100003	342123
100003	451231
.....	.....
898654	56432

Figuur 1.9

Als je de gegevens uit de tabel *Boeking* gesorteerd op klantnummer wenst te tonen, wordt gebruikgemaakt van dat indexbestand. In het indexbestand wordt de eerste rij geselecteerd (klantnummer 100). Het rijnummer 654321 verwijst naar het rijnummer in de tabel *Boeking*. Dus de eerste boeking van klantnummer 100 (laagste klantnummer) is te vinden in rij 654321 in de tabel *Boeking*. Die rij in de tabel *Boeking* kan dan getoond worden. Het opzoeken van die rij duurt ongeveer 10 ms, dus dat is geen probleem. Vervolgens worden uit de tabel *Boeking de* rijnummers 765431, 2342, enzovoort getoond. De informatie wordt dan getoond op volgorde van klantnummer. Er kunnen meerdere indexbestanden per gegevensgroep zijn. Bij de tabel *Boeking* zou nog een indexbestand kunnen bestaan op volgorde van *Hotelnr* of *Checkin*. De indexbestanden worden door de database automatisch onderhouden! Als je een nieuwe boeking toevoegt, wordt automatisch het indexbestand bijgewerkt.
Die indexbestanden zijn *noodzakelijk* bij de relaties tussen de gegevensgroepen. In het voorbeeld van een miljoen rijen zou het ongeveer 2:45 uur duren om alle boekingen van een klant te vinden. Maar door gebruik te maken van het indexbestand van de tabel *Boeking* kunnen de boekingen van die klant snel gevonden worden. Aangezien in het indexbestand, zie figuur 1.9 de klantnummers op volgorde staan, kan slim gezocht worden. Er wordt door de database binair gezocht. Bij *binair zoeken* wordt het indexbestand iedere keer in twee delen verdeeld. Het indexbestand heeft net zo veel rijen als de basistabel, dus ook een miljoen. De helft van een miljoen rijen is 500.000. De eerste rij die geselecteerd wordt in het indexbestand is rijnummer 500.000. Stel dat daar klantnummer 651237 en rijnummer 875398 staan. Wij zoeken klantnummer 100003. Aangezien klantnummer 651237 groter is dan 100003, klantnummers liggen op volgorde, zit ons zoekklantnummer in de bovenste 500.000 rijen. Dus met één zoekactie heb je nog maar 500.000 rijen over. Vervolgens wordt van die bovenste 500.000 rijen opnieuw de helft genomen en wordt gekeken in het indexbestand op rijnummer 250.000. Stel dat daar klantnummer 298765 en rijnummer 23238 staan. Op precies dezelfde manier zit

ons zoekklantnummer in de bovenste 250.000; 298765 is groter dan 100003. Na twee zoekacties hebben we nog maar 250.000 rijen in het indexbestand te onderzoeken. In de derde zoekactie wordt weer de helft genomen van bovenste 250.000 rijen, dus wordt gekeken in rijnummer 125.000 van het indexbestand. Als het klantnummer nu kleiner is dan 100003, ligt ons zoekklantnummer tussen rijnummer 125.000 en rijnummer 250.000. Dan wordt weer gekeken op rijnummer 187.500, enzovoort. Bij een miljoen rijen is het aantal zoekacties maximaal 20. Voor de liefhebbers: $^{2}\log x$, waarbij x het aantal rijen is, bepaalt het maximaal aantal zoekacties. Dat is 20 keer 10 ms is 0,2 sec. Dus 2:45 uur wordt gereduceerd naar 0,2 seconden, vandaar dat indexbestanden onmisbaar zijn binnen een database. Zou je een miljard rijen hebben, dan zijn 30 zoekacties nodig om de juiste rij te zoeken, totaal 30 keer 10 ms is 0,3 s. Om een idee te geven hoeveel een miljard is: bij 650.000 boekingen per dag zou het ongeveer vier jaar en drie maanden duren om dat aantal te halen.

1.5 Big data

Bedrijven, instellingen en organisaties verwerken hun bedrijfsvoering in relationele databases. De gegevens in een relationele database liggen gestructureerd opgeslagen in tabellen. Het opvragen van informatie uit de database vindt plaats met de vraagtaal SQL (Structured Query Language). In hoofdstuk 3 wordt dat verder toegelicht met Microsoft Access en in hoofdstuk 4 wordt specifiek SQL toegelicht. Voor het overgrote deel van de bedrijven, instellingen en organisaties voldoet de relationele database. Ook bij veel data (big data) kan een relationele database prima voldoen, zie de vorige paragraaf. Maar er zijn bedrijven waarbij de database met tienduizenden tegelijkertijd wordt benaderd. Bijvoorbeeld de Amerikaanse supermarktketen Walmart verwerkt per uur een miljoen transacties. Amazon verwerkt meer dan 500.000 transacties per seconde, met pieken van een miljoen transacties per seconde! Dat is wel heel erg veel data en het overschrijdt de grenzen van een op SQL gebaseerde database. Daarvoor moest een andere techniek worden bedacht, ook wel *NoSQL* genoemd. SQL is geoptimaliseerd voor gestructureerde gegevens. Maar door met name het internet worden ook zeer veel *ongestructureerde gegevens* vastgelegd. Ongestructureerde gegevens hebben geen vaste indeling, denk aan een stuk tekst, Facebookbericht, Twitterbericht, LinkedIn, Snapchat, e-mail, foto, video of een internetpagina. Het is niet te vatten in de 'traditionele' tabel met een rij- en kolomstructuur. Vanzelfsprekend zijn dit ook duizelingwekkende hoeveelheden data. Op Facebook worden per dag zo'n 300 miljoen foto's geüpload en vijf miljard stukjes content gedeeld. Die ongestructureerde gegevens kunnen niet verwerkt worden door op SQL gebaseerde databases. De eerste die met dat probleem te maken kreeg was Google. De gegevens van websites zijn ongestructureerd, de hoeveelheid gegevens is gigantisch en bovendien moeten miljoenen gebruikers tegelijkertijd afgehandeld worden. Geen computer ter wereld die dat had kunnen verwerken. De oplossing die Google bedacht heeft, is het verdelen van de gegevens over veel computersystemen (servers) die parallel de gegevens verwerken. Zo hoeft een goedkope server alleen de toegewezen gegevens te verwerken; vele handen maken licht werk. Komt er meer data, dan kunnen eenvoudig extra servers ingeschakeld worden. Ook het probleem van de vele miljoenen gebruikers die tegelijkertijd informatie opvragen is hiermee opgelost, als maar voldoende servers