

# Inleiding databases

ICT.P.DBO.V18



Reader bij de module Databases  
Propedeuse HBO-ICT

Auteurs: Robbert Menko,  
Bert Wimmenhove

HBO-ICT  
November 2018

## Woord vooraf

Deze reader is je wegwijzer voor het zelfstudiegedeelte van de module databases. Het onderwerp wordt in deze module in 6 weken behandeld en bouwt voort op de kennis die is opgedaan in de module SQL. Een deel van die kennis zal hier kort worden herhaald.

## Inhoudsopgave

<b>Woord vooraf .....</b>	<b>2</b>
<b>1 College 1 – Relationale databases .....</b>	<b>5</b>
1.1 Definities.....	5
1.2 Integriteit van relationele databases en regels.....	6
1.2.1 Entiteit-integriteit .....	6
1.2.2 Referentiële integriteit .....	6
1.2.3 NOT NULL-regel.....	7
1.2.4 Unicitéitsregels.....	7
1.2.5 Andere regels.....	7
1.3 Implementatie van regels in het DBMS .....	8
1.4 Opdrachten .....	9
<b>2 College 2 – Ontwerpproblemen .....</b>	<b>10</b>
2.1 Ieder soort 'ding' zijn eigen tabel .....	10
2.2 De Boyce-Codd-regel .....	14
2.2.1 Afhankelijkheden van een deel van de sleutel.....	15
2.2.2 Afhankelijkheden van andere attributen.....	15
2.3 Opdracht .....	16
<b>3 College 3 – Fysiek ontwerp .....</b>	<b>17</b>
3.1 Performance .....	18
3.1.1 Indexen.....	19
3.1.2 Optimizer .....	20
3.1.3 Artificiële sleutels.....	22
3.1.4 Query-aanpassingen .....	22
3.2 Beveiliging.....	22
3.2.1 Autorisatie.....	22
3.2.2 Views.....	24
3.3 Opdrachten .....	24
<b>4 College 4 – Regels in de database .....</b>	<b>25</b>
4.1 Gedragsregels in de database.....	25
4.2 Procedures en functies.....	25
4.3 Triggers .....	26
4.4 Opdrachten .....	27
<b>5 College 5 – Storage engines en NoSQL-databases.....</b>	<b>28</b>
5.1 Storage engines .....	28
5.2 NoSQL-databases.....	29
5.3 Opdrachten .....	30
<b>6 College 6 – Afronden .....</b>	<b>31</b>
6.1 Transacties.....	31
6.2 Proeftoets .....	33
6.3 Opdrachten .....	33
6.4 Verder lezen .....	33
<b>7 Bibliografie.....</b>	<b>34</b>



# 1 College 1 – Relationale databases

Tijdens deze eerste week maak je kennis met databases. Databases zijn een belangrijk onderdeel van informatiesystemen. In principe omvat ieder informatiesysteem in ieder geval twee onderdelen: een gedragscomponent en een informatiecomponent. Kort door de bocht leidt de gedragscomponent tot de applicaties in het informatiesysteem en de informatiecomponent tot de databases. We zullen in latere modules zien dat de werkelijkheid iets gecompliceerder is.

## 1.1 Definities

De module databases gaat met name over relationele databases. Een relationele database is een door een relationeel databasemanagementsysteem (RDBMS) beheerde verzameling gestructureerd opgeslagen gegevens.

Relationele databases zijn in de jaren late jaren '70 en jaren '80 van de vorige eeuw ontwikkeld op basis van een artikel geschreven door Codd. De daarin beschreven theorie was een reactie op bestaande databasetechnologie van met name netwerk- en hiërarchische databases (Codd, 1970). Hoewel relationele databasesystemen sindsdien al vaak dood zijn verklaard, moet vastgesteld worden dat ze de markt tot op de dag van vandaag beheersen.

Een database voldoet in ieder geval aan de eis dat gegevens moeten kunnen worden opgevraagd, opgeslagen, gewijzigd en verwijderd.

Een relationele database voldoet daarnaast aan de volgende voorwaarden:

- gegevens hebben een eenduidige definitie; die eenduidige definitie maakt het mogelijk dat ze op basis daarvan door verschillende programma's (tegelijktijd) kunnen worden gebruikt;
- elk data-element is toegankelijk door de combinatie van de primaire sleutel, de tabelnaam en de kolomnaam;
- de samenhang tussen gegevens is geheel in de database gedefinieerd, los van de programma's die ze gebruiken;
- de databasebeschrijving of -catalogus is op logisch niveau opgeslagen in tabellen en toegankelijk met SQL;
- programma's krijgen de gegevens uit de database aangeboden en kunnen deze bewerken volgens een zogenaamde logische weergave (de tabelvorm), welke is losgekoppeld van de opslag van de gegevens (de technische weergave); dit maakt het mogelijk om wijzigingen in de manier van opslaan aan te brengen zonder dat de genoemde programma's hier hinder van ondervinden; dit noemen we 'fysieke gegevensafhankelijkheid';
- data-integriteit moet in de database kunnen worden opgenomen.

Een RDBMS is het geheel van software waarmee relationele databases kunnen worden gedefinieerd, gebruikt, onderhouden en beheerd. Een RDBMS kan in ieder geval:

- meerdere verzoeken tegelijkertijd afhandelen zonder dat applicaties en/of gebruikers elkaar daarbij (te veel) in de weg zitten;
- ervoor zorgen dat applicaties en gebruikers alleen toegang tot die data hebben waarvoor ze geautoriseerd zijn;

- regels afdwingen die gelden voor de inhoud van de database;
- fouten bij de verwerking en daarmee gepaard gaande ongewenste wijzigingen van de inhoud ongedaan maken;
- meerdere wijzigingen van één applicatie/gebruiker in behandeling nemen zonder dat dit direct leidt tot foutmeldingen en zonder dat dit direct zichtbaar is voor andere gebruikers (transactiemechanisme).

Wij gebruiken bij deze module MariaDB als voorbeeld van een RDBMS. De top 4 meest gebruikte DBMS'en zijn Oracle, Microsoft SQL-server, PostgreSQL en MySQL, allemaal relationele systemen (Solid-IT, 2018). We zullen daarnaast een blik werpen op nummer 5, MongoDB, een niet-relatieveel DBMS (NoSQL).

### 1.2 Integriteit van relationele databases en regels

Een RDBMS dient in de eerste plaats om gegevens te kunnen opvragen, opslaan, wijzigen en verwijderen. Tegelijkertijd moet een RDBMS er ook voor zorgen dat die gegevens integer zijn en blijven. Volgens de vereisten waar een RDBMS aan moet voldoen, moeten de elementen benodigd voor die data-integriteit in de database zelf kunnen worden opgenomen. We zullen eerst beantwoorden wat data-integriteit inhoudt, vervolgens welke integriteitsregels er zijn en ten slotte hoe deze in de database worden ondersteund.

#### Data-integriteit

De informatie in een database moet een zo goed mogelijke weergave zijn van de werkelijkheid die we willen vastleggen. Dat noemen we de integriteit van de database. Als een database integer is geeft deze een getrouwe weergave van die werkelijkheid middels de gegevens die erin opgeslagen zijn.

Het zijn echter niet alleen de gegevens die opgeslagen zijn die de integriteit bepalen, het zijn ook de regels waaraan die gegevens moeten voldoen die daarin een beslissende rol spelen. Als we bijvoorbeeld een artikel alleen verkopen als daar altijd een betaling van de klant tegenover staat, zullen we dat ook in de database moeten afdwingen.

#### 1.2.1 Entiteit-integriteit

De eis dat elke entiteit (of rij in een tabel) in de database uniek moet kunnen worden geïdentificeerd, realiseren we met behulp van de primaire sleutel. Dat is een attribuut of een combinatie van attributen die elke entiteit een unieke identiteit verschaft. Dat heeft consequenties voor dat attribuut of combinatie van attributen. Die mag of mogen geen NULL-waarde(n) bevatten.

#### 1.2.2 Referentiële integriteit

De regel stelt dat elk ding zijn eigen tabel heeft, maar die gegevens die we dan verdelen over meerdere tabellen stonden niet voor niets bij elkaar. Er zijn verbanden tussen die gegevens die verloren dreigen te gaan door het opsplitsen in tabellen. Dit verlies van informatie kan voorkomen worden door bij elke splitsing (referentiële) verwijzingen tussen de tabellen op te nemen. Dat gebeurt altijd met behulp van attributen in de tabellen. We hebben in de cursus SQL al gezien dat dat altijd een primaire sleutel betreft aan de één-kant (parent) en een kopie daarvan als niet-sleutelattribuut in de andere tabel (child), of een primaire sleutel aan de één-kant

(parent) en een kopie daarvan als deel van de primaire sleutel in de andere tabel (child). De referentiële integriteitsregel houdt een verbod in op loze verwijzingen. Aan de hand van Figuur 7 zullen we dit kort illustreren.

### Verwijzingen

Naast referentiële verwijzingen bestaan er nog andere soorten verwijzingen die we in deze module niet expliciet zullen bespreken. Ook die verwijzingen kun je gebruiken om informatie aan je database te onttrekken.

### 1.2.3 NOT NULL-regel

De NOT-NULL-regel stelt dat als een entiteit bestaat ook dat attribuut bekend moet zijn. Van een student moet bijvoorbeeld altijd de naam bekend zijn (student s2001265 heet Aad Donker). In de tabel zie je dan dat die kolom altijd bij elke gevulde regel een waarde bevat. De NOT-NULL-regel geldt altijd bij definitie voor de primaire sleutelattributen in een tabel (volgens de entiteit-integriteitsregel).

### 1.2.4 Uniciteitsregels

Uniciteitsregels worden vertaald als de sleutels in de databasetabellen. Naast de primaire sleutel kunnen er nog meer attributen of combinaties van attributen bestaan in een tabel die uniek zijn voor één entiteit. Naast de postcode-huisnummercombinatie zal ook de provincie-woonplaats-straatnaam-huisnummercombinatie wel uniek één adres aanwijzen. Deze uniciteitsregels kunnen helpen om de database integer te houden door foutieve invoer of onjuiste updates te weigeren. De primaire sleutel is de belangrijkste uniciteitsregel. Andere uniciteitsregels in de tabel noemen we alternatieve sleutels.

### 1.2.5 Andere regels

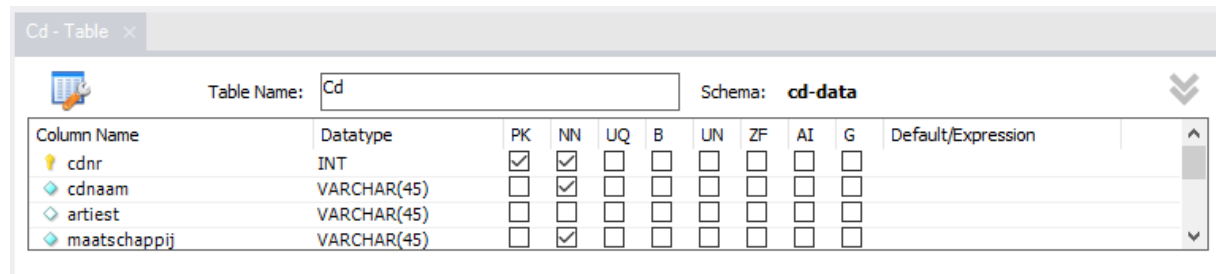
De regels die tot nu toe besproken zijn konden eenvoudig in de datadefinitie van de database worden ondergebracht. Bij de navolgende regels is dat niet het geval. We zullen later bespreken hoe die regels dan wel kunnen worden ingebracht.

Een voorbeeld is de waardenregel. Deze regel geeft aan dat de waarde die een attribuut kan aannemen beperkt is. Zo is het attribuut 'weekdag' bijvoorbeeld beperkt tot de waarden [maandag, dinsdag, woensdag, donderdag, vrijdag, zaterdag, zondag].

Een ander voorbeeld is de aantallenregel. Deze regel geeft aan dat een bepaald attribuut of een bepaalde entiteit slechts een beperkt aantal malen in een tabel mag voorkomen. Zo mocht in het recente verleden in China een ouderpaar slechts één kind hebben. In principe moet dat ook op datbaseniveau afdwingbaar zijn. De uniciteitsregel is een bijzondere vorm van de aantallenregel: een specifiek attribuut mag slechts één keer voorkomen in een specifieke kolom waarop de regel drukt. Niet alle denkbare regels hebben een eigen naam: soms komen ze niet vaak genoeg voor om een eigen naam te rechtvaardigen.

## 1.3 Implementatie van regels in het DBMS

Een beperkt aantal regels kan vrij gemakkelijk tijdens het invoeren van het ontwerp in de workbench ingebracht worden. Deze regels komen echter wel veruit het meeste voor. De workbench genereert dan de daarvoor benodigde SQL-code.

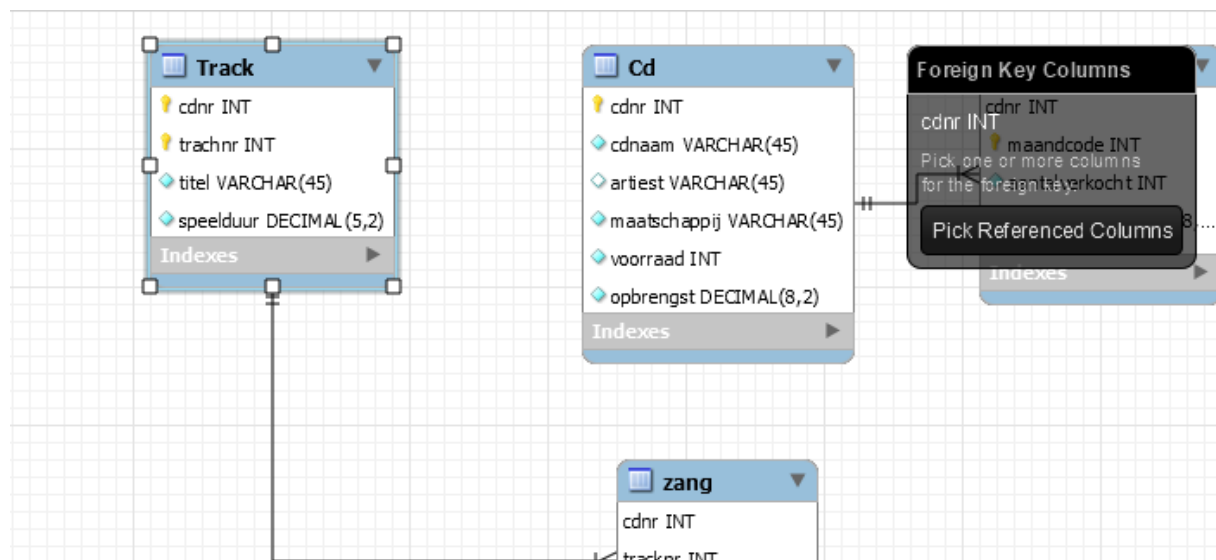


Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
cdnr	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
cdnaam	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
artiest	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
maatschappij	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figuur 1. Regels in de MySQL Workbench

Het datatype geeft als eerste de mogelijkheid om de invoer te beperken tot de gewenste. De kolommen PK (primary key) en NN (NOT NULL) zijn qua gebruik al bekend uit de module Databases en SQL: deze twee garanderen de entiteit-integriteit. Als je PK aanvinkt wordt NN automatisch meegenomen. UQ staat voor Unique key en kan gebruikt worden voor de alternatieve sleutels. Je kunt ook combinaties van attributen uniek maken met behulp van dit mechanisme<sup>1</sup>. Hoewel een Unique key en een uniciteitsconstraint op een subtiele manier van elkaar verschillen zal de werking in de praktijk niet verschillen. Als je zelf het commando voor 'create table' uitschrijft kun je dit verschil wél maken.

De andere kolommen dienen andere doeleinden dan regels: zo dient de kolom AI bijvoorbeeld om aan te geven dat het DBMS de inhoud van het veld zelf moet ophogen bij invoer van een nieuw record.



Figuur 2. Referentiële integriteit in de MySQL Workbench

<sup>1</sup> Selecteer daarvoor de betreffende tabel in de Navigatorlijst, klik op het eigenschappensymbool achter de tabelnaam. Vink UQ aan voor de eerste kolom, selecteer het tabblad Indexes en daarin de betreffende index (altijd met het achtervoegsel \_UNIQUE en vink de 2<sup>e</sup> kolom aan onder 'Index columns'. Als je nu op 'Apply' klikt zie je de SQL-code die de workbench hiervoor genereert.



De referentiële integriteit definiëren we bij het aanbrengen van de relaties: ook dit ken je uit de module 'Databases en SQL'.

Er is voor andere regels geen standaardinstelling aanwezig in de workbench (ook niet in SQL). We vallen dan terug op andere mechanismen om de regel toch te kunnen afdwingen. We zullen later in deze module twee belangrijke hulpmiddelen daarvoor bespreken: triggers en stored procedures. Een aantal andere zullen we niet behandelen omdat MySQL en MariaDB ze niet ondersteunen (user defined types en domains).

### 1.4 Opdrachten

De opdrachten bij dit college worden op de elo gepubliceerd..

## 2 College 2 – Ontwerpproblemen

Het vermijden van redundantie lijkt een centraal thema in de relationele database-theorie. Het is echter niet zo dat redundantie in een relationele database verboden is of zelfs maar ongewenst zou zijn. Het gaat erom de problemen die redundantie veroorzaakt (onbetrouwbare informatie en daarmee informatieverlies), te vermijden. Daarom wordt er altijd eerst een of meerdere ontwerpen (op conceptueel en logisch niveau) gemaakt waarin redundantie uitgebannen is om vervolgens in de fysieke implementatie weer terug te keren, maar dan in een volstrekt beheerste vorm. Redundantieproblemen worden in de relationele-databasetheorie opgelost door te normaliseren. Normaliseren betekent praktisch gezien altijd dat een tabel wordt opgesplitst in twee aparte tabellen. Daarbij mag het verband tussen de onderdelen niet verloren gaan (om informatieverlies te voorkomen), er moeten dan (referentiële) verwijzingen aangemaakt worden tussen de verschillende tabellen. Zo is er een 'normalisatietheorie' ontstaan die in een aantal stappen komt tot een volledig genormaliseerde databasestructuur zonder redundantie. Helaas is deze 'normalisatietheorie' nooit echt geformaliseerd tot één standaard en werkt het bijbehorende stappenplan alleen in standaardsituaties echt naar tevredenheid. De theorie rond normaliseren dient in de praktijk dan met name ook nog als hulpmiddel ter controle van ontworpen structuren: "als je in één keer een goede structuur opzet, met 'elk soort ding zijn eigen tabel', hoef je niet meer te normaliseren of te standaardiseren." (Wiegerink, Bijpost, & de Groot, 2018, p. 152). Voor ons doeleind zijn maar twee regels uit de 'normalisatietheorie' belangrijk:

1. elk soort 'ding' berg je op in zijn eigen tabel
2. elk attribuut in een RDBMS-tabel is functioneel afhankelijk van alleen de volledige primaire sleutel (de Boyce-Codd-regel).

Deze regels bieden ook voldoende steun als een ontwerp onverhoopt toch nog redundantie bevat. We zullen beide regels en de toepassing ervan toelichten met voorbeelden.

### 2.1 Ieder soort 'ding' zijn eigen tabel

Deze module is niet bedoeld voor het ontwerpen van relationele databases of om uit te leggen wat de theorie rond normaliseren precies inhoudt, maar je zult wel geconfronteerd worden met de vraag wat nu een goed databaseontwerp is.

Om hierop antwoord te kunnen geven moet je kijken welke problemen kunnen ontstaan bij het bewaren van gegevens in een relationele database.

In Figuur 3 zie je een informatiebehoefte van een docent weergegeven. In deze vorm kunnen we deze informatie weliswaar opslaan maar dat zou met grotere aantallen problemen opleveren met bijv. het aanbrengen van wijzigingen in de omschrijving bij een module.

## Inleiding databases

Studentnr: S2001265	Coach: HPR01		
Naam: Aad Donker	Naam: Roel de Hoop		
Klas: ICTM2d	Kamer: T5.22		
(indien van toepassing) Studentcoach: S1099538			
Huidige stand van zaken m.b.t. beoordeling onderwijseenheden			
Module-code	Omschrijving	Datum	Beoordeling
BDA1	Bouwen Database-Applicaties 1	11-11-2015	7
BED1	Bedrijfskunde 1	12-11-2015	4
.....	.....	.....	...
.....	.....	.....	...
BDA2	Bouwen Database-Applicaties 2	20- 1-2015	6
ENB	Engels-basis	21- 1-2015	8
PPO1	Persoonlijke Professionele Ontwikkeling 1	22- 1-2015	4
.....	.....	.....	...
BED1	Bedrijfskunde 1	27- 1-2015	5
PPO1	Persoonlijke Professionele Ontwikkeling 1	5 - 2-2015	6
.....	.....	.....	...
.....	.....	.....	...
.....	.....	.....	...

*Figuur 3.* Het studievoortgangsoverzicht (uitgeprint t.b.v. de docent)

De organisatie zou kunnen besluiten om alle informatie waar men mee werkt op te slaan, zoals weergegeven in Figuur 4 of 5. Het oorspronkelijke studievoortgangsoverzicht uit Figuur 3 kan weer gereconstrueerd worden uit deze tabellen (want er mag natuurlijk geen informatie verloren gaan).

studentnr	naam	Klas	coach-id	coach-naam	kamer	student-coach	module-code	omschr.	datum	beoordeling
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	T5.22	S1099538	BDA1	Bouwen DB-Appl.1	11-11-2015	7
							BED1	Bedrijfskunde 1	12-11-2015	4
							.....	.....	.....	...
							BDA2	Bouwen DB-Appl.2	20-1-2015	6
							ENB	Engels-basis	21-1-2015	8
							PPO1	Pers.Prof.Ontw.1	22-1-2015	4
							.....	.....	.....	...
							BED1	Bedrijfskunde 1	27-1-2015	5
							PPO1	Pers.Prof.Ontw.1	5-2-2015	6
							.....	.....	.....	...
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	T4.09		BDA1	Bouwen DB-Appl.1	11-11-2015	6
							.....	.....	.....	...
							BDA2	Bouwen DB-Appl.2	20-1-2011	5
							ENB	Engels-basis	21-1-2011	7
							PPO1	Pers.Prof.Ontw.1	22-1-2011	7
							BED1	Bedrijfskunde 1	27-1-2011	6
S2001401	Viola Glas	ICTM2i	TPE01	Els Tulp	T2.57		BDA1	Bouwen DB-Appl.1	11-11-2010	8
							BED1	Bedrijfskunde 1	12-11-2010	9
							.....	.....	.....	...
							BDA2	Bouwen DB-Appl.2	20-1-2011	9
							ENB	Engels-basis	21-1-2011	6
							PPO1	Pers.Prof.Ontw.1	22-1-2011	8
							.....	.....	.....	...

*Figuur 4.* Het studievoortgangsoverzicht in tabelvorm

### Elementaire gegevens

In een relationele database moeten alle attributen (kolommen in de tabel) elementair zijn. Dat betekent niet dat attributen op geen enkele manier samengesteld kunnen zijn.

Als een naam b.v. is opgebouwd uit voornaam en een achternaam, kun je een attribuut 'naam' kiezen, waar je de hele naam mee aanduidt en later opslaat in één kolom. Hier kun je voor kiezen, als je de naam altijd alleen maar in z'n geheel gebruikt. Als je de voornaam en de achternaam echter ook los van elkaar wilt gebruiken, kun je de naam het beste opsplitsen in de attributen 'voornaam' en 'achternaam' (in dit geval kan een attribuut 'tussenvoegsel' ook verstandig zijn). Het is dus verstandig om vooraf na te gaan of de gebruiker een adres altijd als geheel gebruikt of dat er ook toepassingen zijn waarbij de gebruiker straatnaam en huisnummer los van elkaar wil gebruiken.

De SQL-standaard staat het gebruik van samengestelde **datatypes** in verregaande mate toe: een kolom kan dan zelfs een tabel bevatten (Date & Darwen, 2000; ISO/IEC, 2016). Moderne implementaties van relationele databases ondersteunen dat. De tabel uit Figuur 4 zou misschien wel op deze manier in een database geplaatst kunnen worden.

studentnr	naam	klas	coach-id	coach-naam	kamer	student-coach	module-code	omschr.	datum	beoordeling
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	T5.22	S1099538	PPO1	Pers.Prof.Ontw.1	22-1-2015	5
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	T4.09	S1099536	PPO1	Pers.Prof.Ontw.1	22-1-2015	8
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	T5.22	S1099538	BDA1	Bouwen DB-app	11-1-2015	7
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	T5.22	S1099538	BDA2	Bouwen DB-app	23-1-2015	6
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	T4.09	S1099536	BED1	Bedrijfskunde 1	25-1-2015	8
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	T5.22	S1099538	BED1	Bedrijfskunde 1	25-1-2015	5
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	T5.22	S1099538	ENB	Engels basis	26-1-2015	6
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	T4.09	S1099536	BDA1	Bouwen DB-app	11-1-2015	8
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	T4.09	S1099536	BDA2	Bouwen DB-app	23-1-2015	7
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	T4.09	S1099536	ENB	Engels basis	26-1-2015	7
.....										

Figuur 5. Studievoortgangsgegevens in één tabel.

In deze laatste tabel zie je een aantal zaken opvallend vaak terugkomen. Dit vraagt op zich weliswaar iets meer geheugen- en opslagruimte, maar het echte probleem bij beide tabellen is dat als er iets aangepast moet worden, dat op veel plaatsen moet gebeuren, waardoor er zomaar iets over het hoofd gezien kan worden. Dit leidt dan tot inconsistenties in de database (wat net zo erg is als informatie verliezen).

De oplossing is om deze tabel op te splitsen: de herhaald voorkomende gegevens worden in aparte tabellen geplaatst waarbij geldt dat 'ieder soort ding z'n eigen tabel' krijgt. We gaan in deze module niet in op hoe je dat precies moet doen: daarvoor heb je ontwerpmethoden die je helpen om op conceptueel niveau een ontwerp maken

(Halpin, 2001). Dat opsplitsen mag er echter niet toe leiden dat er informatie uit het oorspronkelijke studievoortgangsoverzicht verloren gaat! Daarvoor nemen we verwijzingen op tussen de tabellen.

We zullen hier uitgaan van de redelijk intuïtieve notie van een 'soort ding', zoals eerder toegepast bij het opstellen van het domeinmodel in de module AFO. Door het opsplitsen in aparte tabellen staat het ontwerp in principe in de eerste normaalvorm: er komen geen herhaalde gegevens meer voor in het ontwerp. Een student, een coach, een module en een beoordeling zijn in principe gemakkelijk herkenbaar. Een eerste oplossing ziet er dan misschien zo uit, maar daarbij zijn er wel een paar zaken die nog aandacht behoeven.

### Student

<u>studentnr</u>	naam	klas	coach-id	coachnaam	studentcoach
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	S1099538
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	S1099536
S2001388	Dora Ziek	ICTM1c	HPR01	Roel de Hoop	S1099540

### Coach

<u>coach-id</u>	kamer
HPR01	T5.22
GLG02	T4.09

### Module

<u>module-code</u>	omschr.
PPO1	Pers.Prof.Ontw.1
BED1	Bedrijfskunde 1
ENB	Engels basis
BDA1	Bouwen DB-app
BDA2	Bouwen DB-app

### Beoordeling

<u>studentnr</u>	<u>modulecode</u>	beoordeling	Datum
S2001265	PPO1	5	22-1-2015
S2001397	PPO1	8	22-1-2015
S2001265	BDA1	7	26-1-2015
S2001265	BDA2	6	11-1-2015
S2001397	BED1	8	23-1-2015
S2001265	BED1	5	23-1-2015
S2001265	ENB	6	24-1-2015
S2001397	BDA1	8	26-1-2015
S2001397	BDA2	7	11-1-2015
S2001397	ENB	7	24-1-2015

*Figuur 6. Genormaliseerde tabellen.*

## 2.2 De Boyce-Codd-regel

In het verleden werd in de normalisatietheorie een drietal stappen (of 6 of meer als je ook BCNV, 4NV, 5NV, etc. meerekent) beschreven waarin tot een genormaliseerde databasestructuur werd gekomen. De Boyce-Codd-regel (officieel BCNV – Boyce-Codd-normaalvorm) vervangt een belangrijk deel van deze stappen. De regel stelt dat elk attribuut in een tabel functioneel afhankelijk moet zijn van alleen de volledige primaire sleutel. We zullen in het vervolg aan de hand van voorbeelden van problemen laten zien hoe dat werkt.

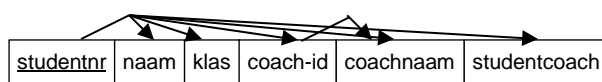
### Functionele afhankelijkheid

In de theorie rond normaliseren speelt het begrip ‘functionele afhankelijkheid’ een belangrijke rol. Een voorbeeld zal dat verduidelijken.

Student

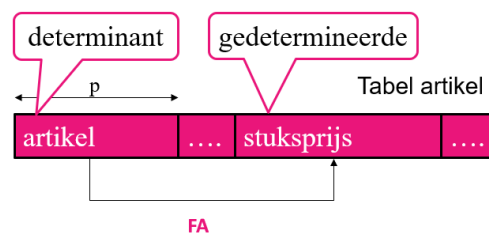
<u>studentnr</u>	naam	klas	coach-id	coachnaam	studentcoach
S2001265	Aad Donker	ICTM2d	HPR01	Roel de Hoop	S1099538
S2001397	Paul Griep	ICTM2a	GLG02	Gea Gul	S1099536

De primaire sleutel voor deze tabel is het studentnr. De theorie stelt dat ieder attribuut in deze tabel volledig functioneel afhankelijk moet zijn van alleen die sleutel. Wat betekent dat? De naam Aad Donker hoort bij de student met studentnr s2001265. Als je een ander studentnr neemt, bijv. s2001397, dan hoort daar een andere naam bij, in dit geval Paul Griep. De naam Aad Donker hangt van geen enkel ander attribuut af: als bijvoorbeeld de klas verandert naar ICTM2e omdat s2001265 van klas verandert heeft dat geen enkele consequentie voor de relatie tussen s2001265 en Aad Donker: s2001265 heet nog steeds Aad Donker. Als je daarentegen S2001265 een andere coach geeft, bijvoorbeeld GLG02, dan moet je ook de coachnaam in dat record veranderen in Gea Gul. Klaarblijkelijk is die coachnaam niet alleen volledig functioneel afhankelijk van het studentnr maar ook van het coach-id.



De pijlen beelden de functionele afhankelijkheid uit. Je kunt ze ook lezen als ‘studentnr bepaalt de naam’. Je ziet nu dat coachnaam niet alleen van studentnr afhangt maar ook van coach-id.

Hieronder zie je de visuele definitie van het begrip functionele afhankelijkheid.



Artikel *determineert* stuksprijs

Stuksprijs *is functioneel afhankelijk van* artikel

### 2.2.1 Afhankelijkheden van een deel van de sleutel

In de tabel Beoordeling in Figuur 6 is de datum vastgelegd waarop de beoordeling is gegeven. Als je goed kijkt zie je dat de datum bij elke module hetzelfde is (in dit voorbeeld!). Als na verder onderzoek mocht blijken dat dit inderdaad de regel is binnen deze organisatie, betekent dat dat de datum alleen maar bepaald wordt door de module en niet door de combinatie van student en module (de volledige primaire sleutel van deze tabel). Iedere keer als nu een rij met een reeds voorkomende module wordt toegevoegd wordt ook dezelfde datum toegevoegd. Het redundantieprobleem is dus nog niet opgelost. De oplossing is dan om dat gegeven (de datum) uit deze tabel te halen en onder te brengen in de tabel van 'zijn eigen ding' Module. Als er niet zo'n tabel aanwezig is moet je een nieuwe tabel maken met als primaire sleutel de unieke identificatie van het ding 'module'.

### 2.2.2 Afhankelijkheden van andere attributen

In de tabel Student in Figuur 6 worden ook meerdere gegevens van de coach van de student vastgelegd. Iedere keer als Roel de Hoop coach van een student is, worden zowel zijn coach-id als zijn naam bij die student vastgelegd. Ook dit is redundantie. Oplossing: verwijder één van beide attributen uit de tabel. Daarbij moet je er zeker van zijn dat het achterblijvende attribuut uniek verwijst naar de coach. In dit geval kun je het attribuut coachnaam onderbrengen in de tabel Coach. Als er niet zo'n tabel aanwezig is moet je een nieuwe tabel Coach maken, waarbij je het attribuut coach-id als primaire sleutel gebruikt: dat is in dit geval immers de unieke verwijzing naar een coach.

Coach

<u>coach-id</u>	coachnaam	kamer
HPR01	Roel de Hoop	T5.22
GLG02	Gea Gul	T4.09

Module

<u>module-code</u>	omschr.	datum
PPO1	Pers.Prof.Ontw.1	22-1-2015
BED1	Bedrijfskunde 1	23-1-2015
ENB	Engels basis	24-1-2015
BDA1	Bouwen DB-app	26-1-2015
BDA2	Bouwen DB-app	11-1-2015

### Beoordeling

<u>studentnr</u>	<u>modulecode</u>	Beoordeling
S2001265	PPO1	5
S2001397	PPO1	8
S2001265	BDA1	7
S2001265	BDA2	6
S2001397	BED1	8
S2001265	BED1	5
S2001265	ENB	6
S2001397	BDA1	8
S2001397	BDA2	7
S2001397	ENB	7

### Student

<u>studentnr</u>	<u>naam</u>	<u>klas</u>	<u>coach-id</u>	<u>studentcoach</u>
S2001265	Aad Donker	ICTM2d	HPR01	S1099538
S2001397	Paul Griep	ICTM2a	GLG02	S1099536
S2001388	Dora Ziek	ICTM1c	HPR01	S1099540

*Figuur 7. Volledig genormaliseerde tabellen in BCNV*

In de praktijk is dit in de meeste gevallen afdoende. Er zijn nog wel meer normaalvormen (4NV, 5NV, PJNV, etc.) waarmee allerlei minder vaak voorkomende problemen in ontwerpen kunnen worden opgelost. Probleem is wel dat deze onregelmatigheden vaak nauwelijks opvallen in een ontwerp. Alleen gebruik van een methode om een conceptueel model te ontwerpen lost dit probleem echt op (Halpin, 2001).

#### Referentiële integriteit

Aan de hand van Figuur 7 is goed te zien wat er gebeurd is bij het opsplitsen van tabellen. De oorspronkelijke tabel is opgesplitst in vier tabellen. Zo is de coach afgesplitst van de student. Om de informatie uit de oorspronkelijke tabel te kunnen reconstrueren is het coach-id in de tabel Student opgenomen. Dit coach-id verwijst naar het coach-id in de tabel Coach. Het verbod op loze verwijzingen stelt dat er in de tabel Student nooit coach-id's mogen voorkomen die niet bestaan in de tabel Coach.

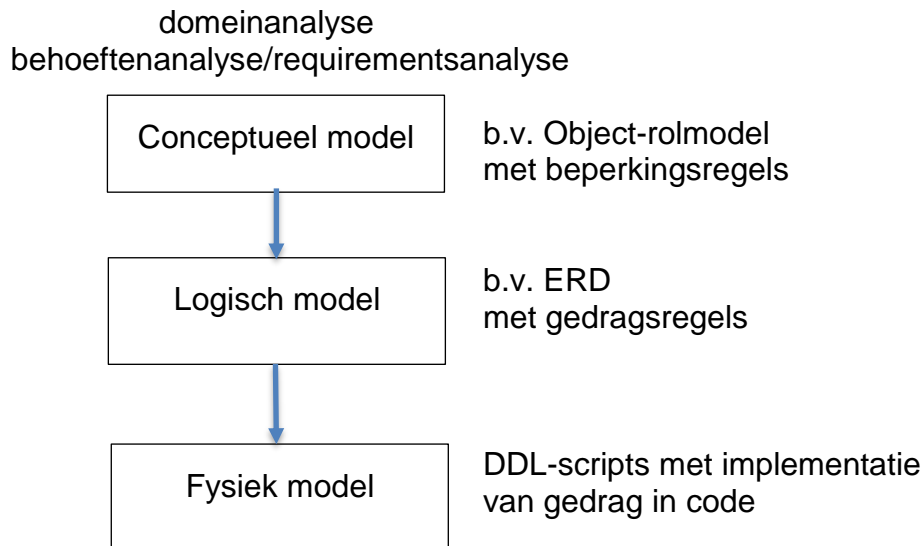
## 2.3 Opdracht

De opdrachten bij dit college worden op de elo gepubliceerd.



### 3 College 3 – Fysiek ontwerp

Het ontwerp van het conceptuele databaseschema (zie Figuur 9) omvat in principe drie verschillende ontwerpstappen: een conceptuele analyse (tijdens de analysefase van de system lifecycle) en een logisch ontwerp leidt tot een logisch databasemodel (in de vorm van bijvoorbeeld een ERD met gedragsregels). Een fysieke ontwerpstap, waarin rekening wordt gehouden met de fysieke beperkingen van de database en het gebruik ervan, leidt tot een fysiek model. Het fysiek model wordt afgebeeld in DDL-SQL-code en definitie van gedrag in code.



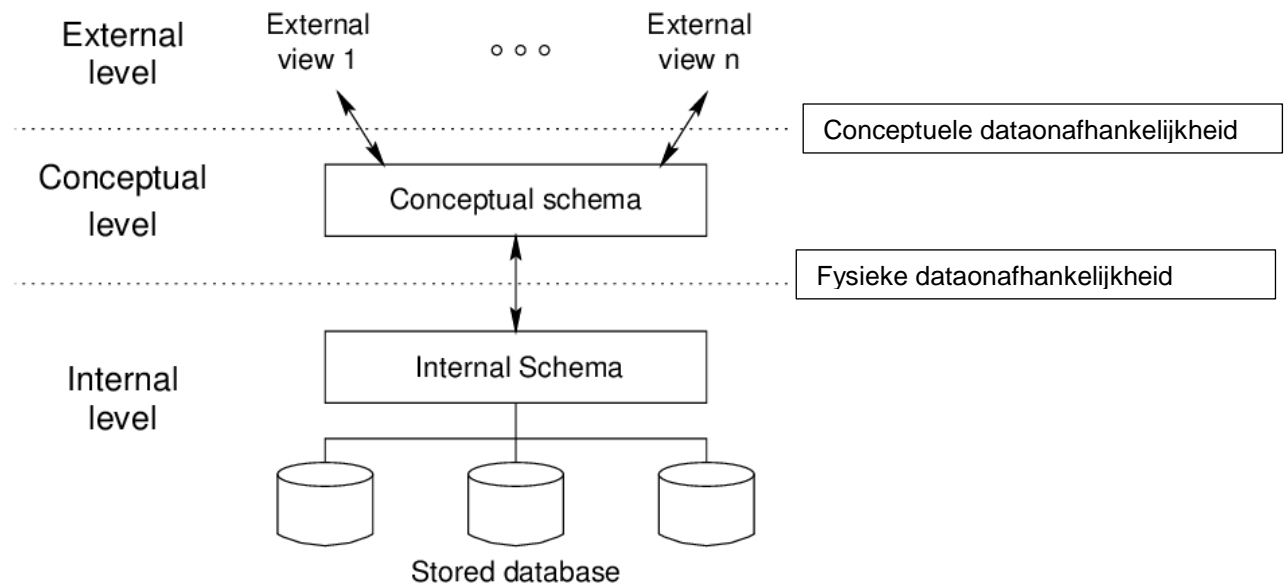
Figuur 8. Ontwerpstappen voor het conceptual schema (zie Figuur 9).

#### Van logisch ontwerp naar fysiek ontwerp

In een wereld met perfecte technologie zouden we het logisch databasemodel (of zelfs het conceptuele model) wellicht wél één-op-één implementeren. In de werkelijke wereld moeten we echter met allerlei beperkingen rekening houden en één van de belangrijkste drijvers is **performance** (Tsichritzis & Lochovsky, 1982). Daarvoor moeten we allerlei maatregelen nemen waarvan we er in dit hoofdstuk een beperkt aantal zullen bespreken. Veel van deze maatregelen leiden ertoe dat de redundantie weer toeneemt. Naast de performanceverhogende maatregelen zullen we dan ook maatregelen moeten nemen die voorkomen dat de geïntroduceerde redundantie tot inconsistenties leidt.

Een andere overweging die leidt tot aanpassingen in het databasemodel is de **beveiliging** van gegevens.

In deze module gaat de aandacht met name uit naar het conceptuele niveau. Het externe niveau komt heel kort aan de orde als we views bespreken in hoofdstuk 3.2.2. Het interne niveau komt kort aan de orde in hoofdstuk 5 als het gaat over de storage engines. Het interne schema beschrijft hoe de data fysiek is opgeslagen op de opslagmedia. Het gaat dan over bestandsstructuren, toegangsmethoden, binaire datastructuren, datacompressie, encryptie, enzovoort. We zullen dit niveau (de storage engine) beschouwen als een black box.



Figuur 9. ANSI/SPARC drieschema-architectuur.

De ANSI/SPARC-architectuur is niet gekoppeld aan relationele databasesystemen maar is een generieke architectuur die voor alle databasesystemen geldt. De architectuur is met name een voorstel om fysieke en conceptuele dataonafhankelijkheid in dbms'en in te bouwen (Tsichritzis & Lochovsky, 1982).

### 3.1 Performance

De parameters die door het opslagsysteem gebruikt worden spelen een grote rol in de performance. Je ziet tegenwoordig steeds meer zogenaamde in-memory-databases, waarbij de data volledig in het interne geheugen van de computer wordt gehouden. Als je bedenkt dat intern geheugen nog steeds een factor 100 tot 1000 sneller is dan extern geheugen kun je wel nagaan wat dit met de performance van de database doet. Hoewel dit een belangrijk aandachtsgebied is voor performanceverbetering zullen we deze hier niet verder bespreken omdat dit niet binnen de scope van deze module valt.

Een ander belangrijk aandachtsgebied hangt samen met het gebruik van de gegevens in samenhang en hoe deze gebufferd worden in het geheugen. Overwegingen met betrekking tot buffergrootte, prefetchingmethoden en vervangingsalgoritmen spelen daarbij een rol. Ook dit onderdeel zullen we hier verder buiten beschouwing laten.

Een volgend belangrijk aandachtsgebied voor performanceverbetering is de selectie en het gebruik van toegangspaden in het systeem. Toegangspaden zijn opslagstructuren (indices of indexen) die de inhoud van een logisch object koppelen aan het gerelateerde fysieke record in de database. Het probleem is welke toegangspaden je moet onderhouden en hoe je zo goed mogelijk gebruikmaakt van de toegangspaden.

Een laatste aandachtsgebied is de manier waarop een query wordt uitgevoerd. Er is al veel onderzoek gedaan naar query-optimalisatie. Resultaat daarvan is dat de ingebouwde query optimizers veel query's zelfstandig kunnen optimaliseren. Dat deze optimizers daartoe veel statistische data genereren voor eigen gebruik is een bijkomend effect. MariaDB bewaart deze data in een systeemdatabase 'performance\_schema'.

### 3.1.1 Indexen

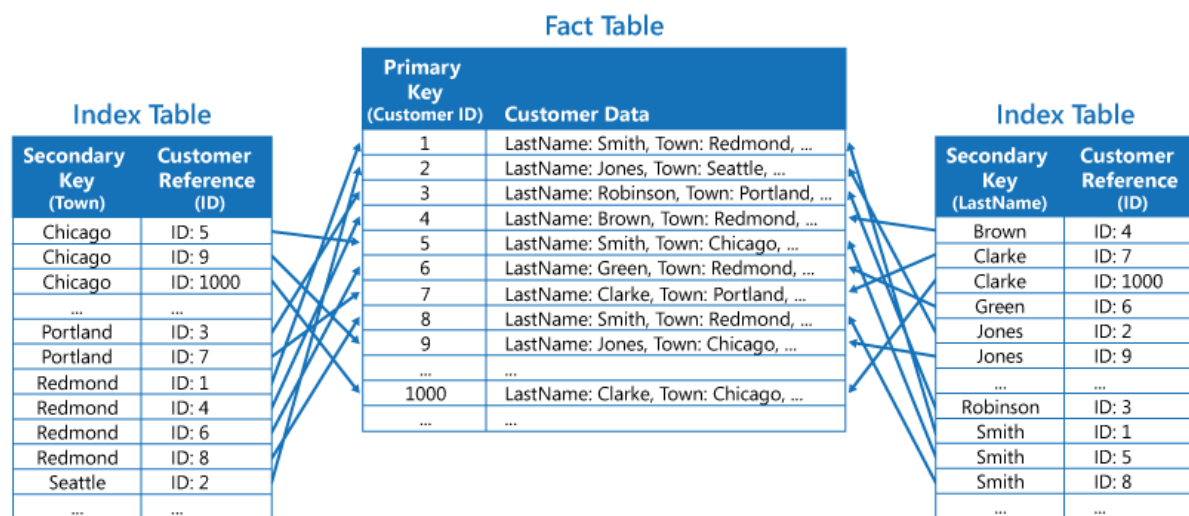
Een database-index is een opslagstructuur die tot doel heeft selectiebewerkingen op een database te versnellen. Een index vermindert het aantal leesbewerkingen dat nodig is om een of meerdere database-records te vinden. Zo wordt voorkomen dat een zogeheten full table scan moet worden gedaan, waarbij alle records in de tabel sequentieel moeten worden doorlopen.

Een database-index kun je qua gebruik (maar niet qua structuur) vergelijken met een index in een boek; door gebruik van de index hoef je niet alle pagina's door te lezen om een bepaald onderwerp te vinden. Als je bijvoorbeeld in Figuur 10 de klanten moet vinden die in Redmond wonen moet je de hele Fact table record-voor-record lezen om die te vinden. Door gebruik te maken van de indextabel waarin gesorteerd is op plaats kun je zeer efficiënt de betreffende records uit de Fact table halen. Er zijn twee algemene voorwaarden te formuleren waaronder het gebruik van indexen zinvol kan zijn:

- zeer grote tabellen waarop veel (aparte) leesbewerkingen moeten worden uitgevoerd m.b.v velden anders dan de primaire sleutel en weinig inserts en/of deletes;
- als er weinig records gevonden zullen worden bij een leesbewerking. Als een kolom bijvoorbeeld de geslachtsaanduiding m/v bevat is het weinig zinvol daarop een index te plaatsen. De overhead van het onderhouden van de index gaat dan zwaarder tellen dan het voordeel.

Indexgebruik heeft dus ook nadelen. In de eerste plaats moet de index worden bijgewerkt als er records worden toegevoegd of verwijderd. Als dat heel veel gebeurt, wordt het systeem zelfs trager dan zonder index. Als dat niet zo is zal er geen merkbaar prestatieverschil zijn voor de meeste hedendaagse databasesystemen, omdat die hun indexen zullen bijwerken op het moment dat het systeem minder zwaar wordt belast.

Bij het aanmaken en configureren van indexen dient altijd een goede afweging te worden gemaakt tussen serverbelasting en snelheidswinst.

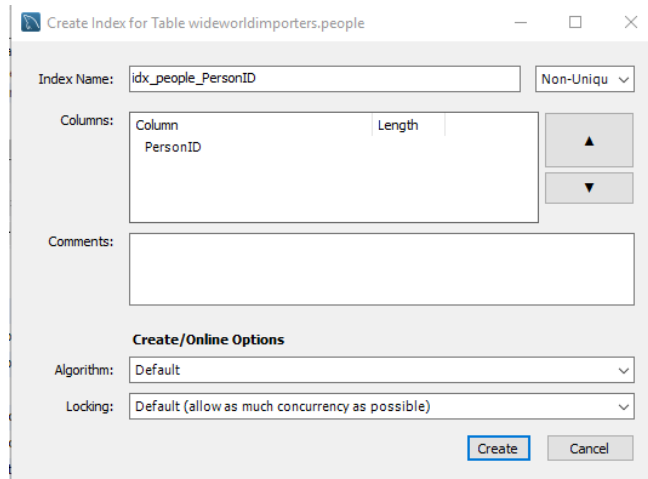


Figuur 10. Gebruik van indexfiles (Microsoft, 2018).

## Inleiding databases

Er bestaan veel verschillende soorten indexen: we zullen ons daar in deze module niet druk over maken. We maken gebruik van de indexen zoals die standaard worden aangeboden.

De MySQL Workbench biedt de mogelijkheid om op kolomniveau indices aan te maken door een rechtermuisklik op de betreffende kolom te geven.



Figuur 11. Het aanmaken van een index in de MySQL-workbench

### 3.1.2 Optimizer

Als je een SQL-query invoert, wordt dat statement door de SQL-interpreter uitgevoerd. De interpreter bestaat in essentie uit twee onderdelen, een parser en een optimizer. De parser checkt of het statement correct is, de optimizer bepaalt hoe de query daadwerkelijk wordt uitgevoerd. Dat laatste moet resulteren in de kortst mogelijke uitvoeringstijd met gebruik van zo min mogelijk middelen.

Hoe snel een query kan worden beantwoord hangt van een aantal zaken af:

- hoeveel records van extern geheugen opgehaald moeten worden
- hoeveel records in intern geheugen gehouden moeten worden
- de tijd die de processor nodig heeft om de bewerkingen uit te voeren.

De optimizer zal proberen zo min mogelijk records te verwerken en zo min mogelijk records in het geheugen te houden. Dat betekent dat de optimizer allerlei technieken moet toepassen om dit te bewerkstelligen. Zo worden bijvoorbeeld de voorwaarden uit de where-clausule zoveel mogelijk uitgevoerd voordat er een join wordt gemaakt, wordt er gebruik gemaakt van de aanwezige indexen, etc. Dit resulteert in een execution plan dat voor elke query wordt gemaakt voor uitvoering ervan. Deze queryplannen worden bewaard en indien mogelijk hergebruikt. Optimizers gebruiken twee werkwijzen die elkaar aanvullen:

- op basis van regels die van tevoren zijn vastgelegd
- op basis van statistische gegevens over de uitvoering van query's; de optimizer van MariaDB heeft hiertoe de beschikking over een eigen database: Performance\_schema<sup>2</sup>.

<sup>2</sup> MySQL 8.0 volgt weliswaar een iets ander pad maar er zijn meer dan genoeg overeenkomsten om de optimalisatiedocumentatie van MySQL te gebruiken (MySQL, 2018).

## Inleiding databases

Je kunt het execution plan inzien door in de CLI het commando EXPLAIN voor een select-query te plaatsen (EXPLAIN werkt niet voor andere query-typen).

Query 1 x wideworldimporters.buyinggroup... wideworldimporters.buyinggroups wideworldimporters.stockitems wi

```
1 SELECT customername FROM customers
2 JOIN invoices USING (customerid)
3 JOIN orders USING (orderid)
4 WHERE isundersupplybackordered = 1
5
```

Tabular Explain

id	select_t...	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	customers	index	PRIMARY	UQ_Sales_Cust...	102		663	Using index
1	SIMPLE	invoices	ref	FK_Sales_Invoices...	FK_Sales_Invoi...	4	widew...	54	Using where
1	SIMPLE	orders	eq_ref	PRIMARY	PRIMARY	4	widew...	1	Using where

Result 21 x Read Only

Figuur 12. Het execution plan in de MySQL-workbench.

De workbench biedt de explain-output zonder dat je zelf expliciet het commando moet geven zoals te zien in Figuur 12. Met behulp van het chevron-teken kun je op de plaats van het outputvenster het executionplan ophalen. Explain toont altijd 10 velden.

**Id** is een identifier en heeft verder geen betekenis. Per tabel/subquery wordt er een rij getoond.

**Select\_type** kan de waarden SIMPLE (een tabel of een JOIN), PRIMARY (de eerste SELECT in een UNION of subquery), UNION, UNION RESULT of bij gebruik van subquery's DEPENDENT UNION, DEPENDENT SUBQUERY, DERIVED of UNCACHEABLE SUBQUERY bevatten.

**Table** geeft de tabelnaam of het alias.

**Type** geeft de zogenaamde data access method. 'all' is de zogenaamde full table scan. Alle andere methoden maken gebruik van de indexen (en zijn daarmee vrijwel altijd beter). 'index' is een full index scan. 'range' is een gedeeltelijke index scan (toegepast bij <, <=, >, >=, IS NULL, BETWEEN en IN). 'index\_subquery' is een subquery met een niet-unieke index op een tabel. 'unique subquery' is een subquery met gebruik van de primaire sleutel of een uniek veld van één tabel. 'index\_merge' als er meer dan één index wordt toegepast. **Extra** toont in dat geval meer informatie (sort\_union, intersection of union). 'ref\_or\_null' als er op niet-unieke indexen wordt gejoined of gezocht en als er NULL-waarden in het resultaat opduiken, waarvoor een extra werkgang nodig is. 'fulltext' als er door een tekst gescand moet worden. 'ref' als er op niet-unieke indexen wordt gejoined/gezocht of als geïndexeerde velden worden vergeleken met =, !=, <, <=, >, >=. Dit is de beste toegangsmethode voor niet-unieke velden. 'eq\_ref' als er op unieke indexen wordt gejoined/opgezocht. 'const' als er op een waarde wordt gezocht en er gebruik gemaakt wordt van een primaire of unieke sleutel. Dit is de snelste toegangsmethode. Ook 'system' is snel: systeemtabellen bevatten maar één waarde (bijvoorbeeld de tijd).

**Possible\_keys, key, key\_len** en **ref** hebben betrekking op het gebruik van de indexen. Hier kun je zien of de bestaande indexen ook daadwerkelijk worden gebruikt. **Ref** laat zien wat er vergeleken wordt, een veld of een 'const'.

**Rows** laat zien hoeveel rijen er ongeveer gescand zijn.

**Extra** kan verschillende (goede, slechte of neutrale) boodschappen bevatten; een aantal daaruit: ‘full scan on NULL key’ duidt op een subquery zonder gebruik te maken van een index (slecht); ‘using filesort’ duidt op een extra werkgang om de data te sorteren (slecht); ‘temporary’ duidt op het gebruik van een tijdelijke tabel (slecht); ‘open\_full\_table’ duidt op poging alle tabelbestanden open te houden (heel slecht).

### 3.1.3 Artificiële sleutels

De performance van query's (met name joins) heeft ook erg te lijden onder het gebruik van lange sleutels (for obvious reasons) en alfanumerieke sleutels. Vaak zie je dan ook dat lange, natuurlijke sleutels uit het ontwerp bij het aanmaken van de database vervangen (of liever: aangevuld) worden door artificiële sleutels (welke je al of niet door het DBMS kunt laten genereren). De oorspronkelijke, natuurlijke sleutels worden in deze gevallen alternatieve sleutels en kunnen een rol spelen bij het integer houden van de database. Het is echter niet bij definitie dat artificiële sleutels altijd beter zouden zijn: in elk concreet geval zal een eigen afweging gemaakt moeten worden omdat het gebruik van artificiële sleutels ook nadelen kent!

### 3.1.4 Query-aanpassingen

Bij het oplossen van een probleem is vaak meer dan één oplossing mogelijk:

- een keuze tussen een oplossing met subselect en een join
- een keuze tussen een subselect met IN en met EXISTS
- een keuze tussen DISTINCT en GROUP BY
- een keuze voor het al dan niet gebruik van functies

Inner joins kan de optimizer het beste verwerken, subselects en outer joins vormen vaker een probleem. Ook de HAVING-clausule vormt een probleem omdat de voorwaarden pas op het laatst kunnen worden toegepast. De EXISTS is sneller dan een query met IN omdat de bewerking stopt na het vinden van het eerste record (Wiegerink, Bijpost, & de Groot, 2018).

## 3.2 Beveiliging

Een dbms moet in staat zijn om de data die het beheert te beschermen tegen ongeoorloofd gebruik en tegen fysiek falen van het databasesysteem.

Ongeoorloofd gebruik kan voorkomen worden door gebruik te maken van een autorisatiesysteem met gebruikersrollen, wachtwoordbeveiliging en autorisatie om bepaalde handelingen te verrichten op bepaalde delen van de database. Een aanvullend mechanisme om ongeoorloofd gebruik tegen te gaan is het view-mechanisme: hiermee is verfijnde autorisatie mogelijk. Beide onderwerpen worden hieronder besproken.

Fysiek falen van databasesysteem kan weliswaar niet voorkomen worden maar de gevolgen kunnen wel ondervangen worden door een uitgekiend systeem van logbestanden, replicatie en backup- en recoverytechnieken. Dit onderwerp zullen we in deze module niet behandelen: dat vergt een cursus op zich. Online is hierover genoeg terug te vinden.

### 3.2.1 Autorisatie

Een van de beveiligingsmethoden is gebaseerd op een autorisatiesysteem met rollen en privileges. Role based access is in omgevingen met veel gebruikers een ‘must’ en voor beveiliging een standaard (NIST, 2018).



## Inleiding databases

De database administrator (DBA<sup>3</sup>) is in de databaseomgeving de superuser met alle rechten. In MariaDB heet deze user 'root' met standaard een 'blank' wachtwoord. Als je om te oefenen gebruik maakt van de workbench is dat niet erg, maar in een productieomgeving mag je die natuurlijk nooit zo laten!

De DBA is verantwoordelijk voor het aanmaken en autoriseren van alle rollen en gebruikers. Rollen bundelen een verzameling rechten waardoor je niet meer iedere individuele gebruiker apart rechten hoeft toe te kennen. Dit vereenvoudigt het gebruik en onderhoud van users en de hen toe te kennen rechten/privileges in verregaande mate. In MariaDB noemt men dit accountmanagement (MariaDB(d), 2018).

### Basiscommando's:

```
CREATE [OR REPLACE] USER [IF NOT EXISTS] username IDENTIFIED BY 'password'
```

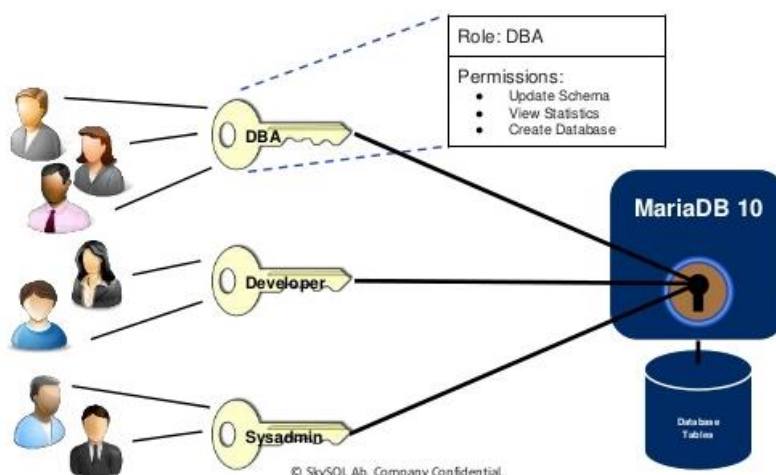
```
CREATE [OR REPLACE] ROLE [IF NOT EXISTS] role
```

```
GRANT priv_type [(column_list)] ON [object_type] priv_level
```

```
TO user [IDENTIFIED [BY [PASSWORD] 'password']
```

Zie ook:

- ALTER USER
- DROP USER
- RENAME USER
- REVOKE
- SET PASSWORD
- DROP ROLE
- SET ROLE
- SET DEFAULT ROLE
- SHOW GRANTS



Figuur 13. Role based access in MariaDB

<sup>3</sup> De DBA is in het algemeen verantwoordelijk voor de autorisatie, performance en beveiliging van het databasesysteem.

### 3.2.2 Views

Een view is een virtuele tabel. Views zijn in essentie 'SELECT-expressies, bewaard onder een eigen naam' (Wiegerink, Bijpost, & de Groot, 2018). Views gedragen zich als tabellen, met het als enig verschil dat die niet opgeslagen worden op extern geheugen en niet altijd te wijzigen zijn.

Hoewel niet de enige toepassing, zijn views een belangrijk mechanisme bij het beveiligen van de database.

Een mooi voorbeeld is het 'information\_schema': de ANSI-standaard voor de verzameling views die informatie verschaft over alle tabellen, kolommen, views, procedures, etc. beheert door het DBMS<sup>4</sup>. We krijgen views en niet de basistabellen te zien omdat de werking van het DBMS afhangt van de integriteit van die basistabellen: één foutje en je DBMS weigert verdere dienstverlening. Je kunt wel veel informatie krijgen over het DBMS. Zo tonen de commando's:

```
USE information_schema;
```

```
SELECT * FROM schemata;
```

alle databases waaronder ook 'mysql' en 'performance\_schema', nog twee systeemdatabases (MariaDB(c), 2018). Het commando SHOW DATABASES geeft alleen databasenames.

We kunnen ook zelf views aanmaken. Met name in de financiële wereld wordt er veel gebruik gemaakt van views om gevoelige informatie af te schermen van gebruikers.

De knowledge base van MariaDB bevat een korte tutorial over het aanmaken en gebruik van views.

## 3.3 Opdrachten

De opdrachten bij dit college worden op de elo gepubliceerd.

---

<sup>4</sup> In de literatuur vind je ook benamingen als data dictionary of systeemcatalogus. Die bevatten meta-data: informatie over informatie.



## 4 College 4 – Regels in de database

Eerder stelden we al vast dat niet alle regels die we op willen leggen aan data via een instelling in de workbench kunnen vastleggen in de database. We moeten dan uitwijken naar andere mogelijkheden die databases daartoe bieden.

Het eerste mechanisme wordt gevormd door CHECK-constraints die op kolomniveau of op tabelniveau kunnen worden toegevoegd aan de datadefinitie van tabellen.

Het tweede mechanisme wordt gevormd door de combinatie van stored procedures en triggers. We zullen beide mechanismen hieronder kort bespreken.

### 4.1 Gedragsregels in de database

MariaDB kent twee manieren om gedragsregels in de definitie van tabellen onder te brengen:

- CHECK (expressie) als onderdeel van een kolomdefinitie
- CONSTRAINT [constraint\_naam] CHECK (expressie) aan het eind van de tabeldefinitie.

Voordat een rij wordt geïnsert of geüpdatet worden alle constraints geëvalueerd in de volgorde waarin ze zijn gedefinieerd. Als een van de constraints niet slaagt wordt de rij niet geüpdatet. De meeste functies kunnen in constraints worden gebruikt.

Een voorbeeld:

```
create table t1 (a int check(a>0), b int check (b> 0), constraint abc check (a>b));
```

In het voorbeeld wordt een tabel t1 gemaakt met twee kolommen a en b, twee kolomconstraints en een tabelconstraint.

### 4.2 Procedures en functies

Er zijn twee basisredenen om stored procedures te willen gebruiken in een database. De eerste is om gedragsregels op een consistente manier te kunnen afdwingen in omgevingen waarin meerdere talen en gebruikers dezelfde bewerkingen moeten uitvoeren.

De tweede reden is beveiliging. Met name banken gebruiken stored procedures om te voorkomen dat applicaties en gebruikers rechtstreeks toegang hebben tot de basistabellen en daar acties uitvoeren die tot gegevensverlies zouden kunnen leiden.

Een stored procedure is een verzameling statements die aangeroepen worden met behulp van een CALL-statement. Een procedure kan in- en outputparameters hebben. Het basiscommando<sup>5</sup> ziet er zo uit:

```
CREATE [OR REPLACE]
  PROCEDURE sp_name ([proc_parameter[,...]])
  routine_body
```

<sup>5</sup> Het volledige commando is iets complexer. Voor onze doeleinden volstaat dit voor nu echter (MariaDB(e), 2018)

## Inleiding databases

```
proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

routine_body:
    Valid SQL procedure statement
```

In het volgende voorbeeld wordt eerst de delimiter gewijzigd om de standaarddelimiter in de body van de procedure te kunnen gebruiken.

```
DELIMITER //
```

```
CREATE PROCEDURE Reset_animal_count()
MODIFIES SQL DATA
UPDATE animal_count SET animals = 0;
//
```

```
DELIMITER ;
```

```
SELECT * FROM animal_count;
```

animals
101

```
CALL Reset_animal_count();
```

```
SELECT * FROM animal_count;
```

animals
0

### 4.3 Triggers

Een trigger is een verzameling statements die uitgevoerd worden als er een bepaalde gebeurtenis (event) plaatsvindt. De gebeurtenis kan een INSERT, UPDATE of een DELETE zijn. De trigger kan worden uitgevoerd voor (BEFORE) of na (AFTER) de gebeurtenis. Een trigger wordt altijd gekoppeld aan een tabel en wordt voor elke rij uit de tabel uitgevoerd. Je kunt meerdere triggers voor één tabel definiëren.

Basiscommando en een voorbeeld:

```
CREATE [OR REPLACE]
  [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
  TRIGGER [IF NOT EXISTS] trigger_name trigger_time trigger_event
  ON tbl_name FOR EACH ROW
  [{ FOLLOWS | PRECEDES } other_trigger_name ]
  trigger_stmt

DROP TABLE animals;
UPDATE animal_count SET animals=0;
CREATE TABLE animals (id mediumint(9) NOT NULL AUTO_INCREMENT,
name char(30) NOT NULL,
PRIMARY KEY (`id`))
ENGINE=InnoDB;

DELIMITER //
CREATE TRIGGER the_moosees_are_loose
AFTER INSERT ON animals
FOR EACH ROW
BEGIN
  IF NEW.name = 'Moose' THEN
    UPDATE animal_count SET animal_count.animals = animal_count.animals+100;
  ELSE
    UPDATE animal_count SET animal_count.animals = animal_count.animals+1;
  END IF;
END; //
DELIMITER ;

INSERT INTO animals (name) VALUES('Aardvark');
SELECT * FROM animal_count;

+-----+
| animals |
+-----+
|      1 |
+-----+

INSERT INTO animals (name) VALUES('Moose');
SELECT * FROM animal_count;

+-----+
| animals |
+-----+
|     101 |
+-----+
```

De 'delimiter' is het teken waarmee een SQL-statement eindigt: standaard is dat de puntkomma. Het wijzigen van de delimiter bij het aanmaken van een trigger is noodzakelijk om de puntkomma in een trigger te kunnen gebruiken om statements van elkaar te kunnen scheiden.

### 4.4 Opdrachten

De opdrachten bij dit college worden op de elo gepubliceerd.

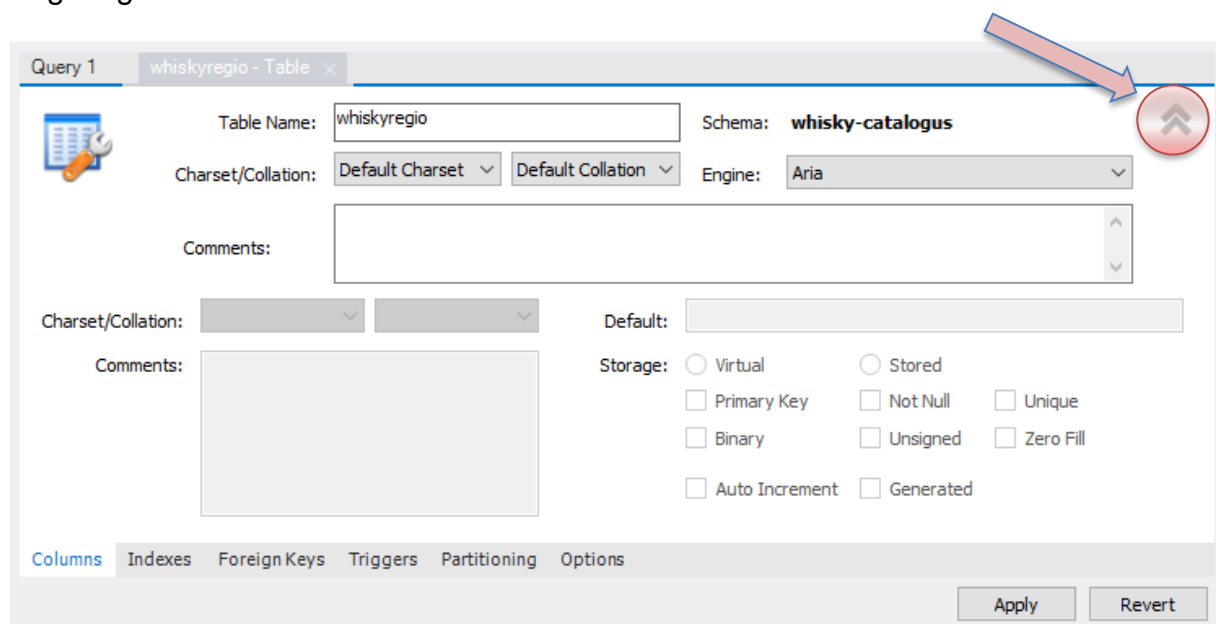
## 5 College 5 – Storage engines en NoSQL-databases

In dit college worden twee aanvullende manieren om data op te slaan behandeld. Eerst wordt gekeken naar de verschillende storage engines van MariaDB en de mogelijkheden die deze bieden.

Vervolgens wordt de aandacht gericht op een ander type database, de NoSQL-database MongoDB. Op het internet vind je goede vergelijkingen tussen de twee (Percona, 2018), hier wordt er daarom een kort overzicht geschetst van de problematiek.

### 5.1 Storage engines

Een database in MySQL (en ook MariaDB) kan meerdere zogenaamde storage engines gebruiken. Je kunt bij de **definitie van een tabel** aangeven welke storage engine gebruikt moet worden.



Figuur 14. De storage engine

Met behulp van het dubbele chevron-teken rechtsboven kun je de eigenschappen van de tabel zichtbaar maken of die van de kolommen.

De default wordt bij de initialisatie van de database of de workbench bepaald. Je kunt dit opvragen door de volgende query uit te voeren:

```
SELECT table_name, engine
FROM Information_schema.Tables
WHERE table_schema='database'
```

Deze query laat zien op welke storage engine elke tabel is gedefinieerd.

Elke engine heeft specifieke eigenschappen welke deze meer of minder geschikt maken voor specifieke situaties (MariaDB(a), 2018). Een niet-volledig overzicht:

- **Archive** storage-engine is bedoeld voor archivering van data.
- **Aria**, MariaDB's verbetering van MyISAM (dat niet wordt onderhouden of verbeterd), gebruikt weinig resources en maakt het kopiëren van data tussen systemen eenvoudiger.
- **ColumnStore** is een massief parallelle gedistribueerde data architectuur, ontwikkeld als schaalbare big-data-oplossing om vele petabytes aan data te kunnen verwerken.
- **CONNECT** maakt toegang mogelijk tot allerlei verschillende soorten tekstbestanden en andere bronnen over een network alsof het gewone MariaDB-tabellen waren.
- **InnoDB** is een goede algemene transactie storage engine. Het is de standaard MySQL storage engine en de standaard MariaDB 10.3 storage engine.
- **MyRocks** kan data beter comprimeren dan InnoDB en genereert minder overhead bij het schrijven van data (ontwikkeld door Facebook).
- **Spider** maakt sharding<sup>6</sup> mogelijk door partitionering over meerdere servers.
- **TokuDB** is een transactionele storage engine geoptimaliseerd voor werkbelastingen die niet in het interne geheugen kunnen worden uitgevoerd; zorgt voor goede compressie<sup>7</sup>'s.

## 5.2 NoSQL-databases

Naast relationele databasesystemen bestaan er nog vele andere vormen van dataopslag. Voor een overzicht zie NoSQLdatabase.org (NoSQL-database.org, 2018). Er is geen formele definitie van NoSQL, maar NoSQL-databases hebben een aantal gemeenschappelijke kenmerken. In de eerste plaats houden ze zich niet aan het relationele model. MongoDB bijvoorbeeld kent geen vast databaseschema, geen tabellen, kolommen of rijen. Een tweede is dat de ontwikkeling van deze databases met name gedreven wordt door de behoefte om grote hoeveelheden data te kunnen verwerken. Daarom leggen deze databases veel nadruk op schaalbaarheid en distributiemogelijkheden. Omdat tabellen, rijen en kolommen ontbreken, maken NoSQL-databases gebruik van andere modelleringsconcepten (NoSQL-database.org, 2018). We nemen voor deze module één specifiek dbms als voorbeeld om geen andere reden dan populariteit. MongoDB staat nr. 5 in het rijtje meest populaire dbms'en en is het eerste niet-relatieve systeem in de lijst (Solid-IT, 2018).

MongoDB is een opensource document-georiënteerde database: een van de soorten NoSQL-databases. Er is geen schema, de documenten worden in de vorm van BSON (binair JSON – JavaScript Object Notation) opgeslagen en de structuur van deze documenten is flexibel. Een document-database bestaat uit zogenaamde collections. Onder een collection verstaan we een verzameling van (min of meer) gelijksoortige documenten. Een collection zou je kunnen vergelijken met een tabel in een RDBMS. De MongoDB-handleiding suggereert om een database-collection te maken voor elk top-level object<sup>7</sup>. Een document zou je kunnen vergelijken met een record in een RDBMS. In tegenstelling tot een RDBMS-record kunnen de veldnamen

---

<sup>6</sup> Sharding = horizontale partitionering van data; vaak aangedragen als reden om voor een NoSQL-oplossing te kiezen ([https://en.wikipedia.org/wiki/Shard\\_\(database\\_architecture\)](https://en.wikipedia.org/wiki/Shard_(database_architecture))).

<sup>7</sup> Zie ter vergelijking de eerste regel uit de normalisatietheorie.

in een document echter dynamisch zijn en kunnen er verschillende soorten gegevens in worden opgeslagen zonder dat de structuur van het document moet worden gewijzigd. Een document bestaat dus uit een verzameling velden (eigenschappen), waarbij een veld bestaat uit een naam en een waarde. In een collection wordt een document geïdentificeerd op basis van een sleutel. Deze sleutel wordt de document-id genoemd en staat altijd in het '\_id'-veld.

De database kan gemakkelijk gedistribueerd worden, de data wordt dan over meerdere computers verspreid om gedistribueerde gegevensverwerking mogelijk te maken. MongoDB biedt geen ondersteuning voor joins en voldoet ook niet aan de ACID-regels want de ondersteuning voor transacties is beperkt (zie hoofdstuk 6).

MongoDB Community Edition is de opensource editie, beschikbaar op Windows, Linux en OS X en wordt middels allerlei diensten commercieel ondersteund door MongoDB Inc, welke ook MongoDB Enterprise Server levert als commerciële editie van MongoDB.

Om snel inzicht te krijgen in wat de mogelijkheden van MongoDB zijn kun je een kijkje nemen op [Tutorials.com](https://www.tutorialspoint.com/mongodb/) (Tutorialspoint, 2018).

Zoals eigenlijk altijd in de halve eeuw dat er RDBMS'en zijn blijft een reactie nooit lang uit. Zo ondersteunt MySQL nu ook native document-opslag (Lefred, 2018), uiteraard middels een storage engine, waarmee het belang van fysieke dataonafhankelijkheid nog maar eens aangetoond wordt.

### 5.3 Opdrachten

De opdrachten bij dit college worden op de elo gepubliceerd.

## 6 College 6 – Afronden

In dit college gaan we kort in op het fenomeen transacties en wordt de proeftoets gemaakt..

### 6.1 Transacties

Opvragingen veranderen niets aan de inhoud van een database. Dat gebeurt wel als je een rij toevoegt aan een tabel, een rij verwijdert uit een tabel of een celinhoud wijzigt in een rij. Om duidelijk te maken waartoe transacties dienen het volgende voorbeeld.

Stel je de database van een bank voor waarin de volgende bewerking moet worden uitgevoerd: Robberts betaald 1000 euro aan Janssens. Hoe moet dit worden uitgevoerd?



rekeningnr	rekeninghouder	saldo
101	Robberts	2356.67
102	Janssens	1356.67

Figuur 15. Bankrekeningen.

```
UPDATE rekening
SET saldo = saldo - 1000
WHERE rekeningnr = 101;
```

```
UPDATE rekening
SET saldo = saldo + 1000
WHERE rekeningnr = 102;
```

Het zal duidelijk zijn dat beide statements altijd óf allebei moeten worden uitgevoerd óf geen van beide moeten worden uitgevoerd. Als door wat voor oorzaak dan ook één van beide niet zou worden uitgevoerd is er óf geld verdwenen óf geld gecreëerd. Geen van beide is voor een bank acceptabel (alleen al omdat de toezichthouder dat niet goed vindt).

De manier om dit probleem aan te vatten is door de statements samen te voegen tot een zogenaamde transactie. In de MySQLworkbench start een transactie met de opdracht om de autocommit uit te zetten: deze staat namelijk standaard aan.

```
SET AUTOCOMMIT = 0
```

```
UPDATE rekening
SET saldo = saldo - 1000
```

```
WHERE rekeningnr = 101;
```

```
UPDATE rekening  
SET saldo = saldo + 1000  
WHERE rekeningnr = 102;
```

```
COMMIT;
```

We kunnen aan het einde van een serie updates nu ook een rollback-opdracht geven. Alle opdrachten die na het SET AUTOCOMMIT = 0 statement zijn gegeven worden dan teruggedraaid. De autocommit is een standaardinstelling van de MySQLworkbench waardoor elk statement direct wordt gecommit. Je ziet een vinkje staan bij het menu-item Query > Auto-Commit Transactions. Verwijder het vinkje en update naar believen een tabel en geef vervolgens de opdracht ROLLBACK. Je zult merken dat je update geen effect heeft gehad. Doe hetzelfde met een COMMIT als afsluiting. Je ziet nu je update terug.

Ons voorbeeld was redelijk eenvoudig. Het transactiemechanisme kan echter ook in andere situaties goede diensten bewijzen. Door dit mechanisme wordt namelijk voorkomen dat elke databaseregels direct na het invoeren van een rij in de database afgaat. Als je bijvoorbeeld een wijn aan de wijndatabase wilt toevoegen moet je ook de referenties naar andere tabellen op orde hebben. In het geval van een wijn moet er een soort, een producent en een gebied ingevuld worden (alle drie NOT NULL). Als daarbij een onbekend gebied zit (nog niet aanwezig in de tabel Gebied) heb je een probleem bij een autocommit: het dbms controleert de referentiële integriteit en constateert dat er een regel overtreden wordt. Uitstellen van de zogenaamde committime maakt het mogelijk om regels tot de COMMIT te overtreden zodat je de tijd krijgt om aan alle regels te voldoen (door bijvoorbeeld meer inserts). Met name bij bulk-inserts wordt dit mechanisme standaard gebruikt.

Nu is transactiebeheer in de praktijk erg lastig omdat er meestal meerdere partijen tegelijkertijd transacties (willen) uitvoeren. Een transactiedatabase moet daarom aan vier eisen voldoen: atomicity, consistency, isolation en durability (ACID). Een database die niet aan deze eisen kan voldoen kan nooit betrouwbaar zijn. De onderdelen van het dbms die daarvoor zorgdragen zijn concurrencycontrol en recoverymanagement.

### **Atomicity**

Eist dat een transactie één geheel is: of in z'n geheel doorgevoerd (COMMIT) of z'n geheel teruggedraaid (ROLLBACK).

### **Consistency**

Transacties moeten de database na afronding altijd weer in een 'geldige' toestand brengen waarbij aan alle integriteitseisen en andere regels is voldaan, ervan uitgaande dat de database bij aanvang van de transactie zich in een 'geldige' toestand bevond. Die eis geldt dus niet tijdens een transactie.

### **Isolation**

Transacties van verschillende databasegebruikers mogen elkaar niet in de weg zitten. Zo mag een artikel in een webshop waarvan er nog maar één voorradig is niet aan twee verschillende klanten tegelijkertijd worden verkocht. Alle dbms'en moeten dit probleem van concurrency op de een of andere manier oplossen. Dit is niet alleen uitsluiten dat transacties elkaar in de weg zitten (dat zou je gemakkelijk kunnen



oplossen door transacties alleen serieel, achter elkaar, plaats te laten vinden – zie je dat al gebeuren bij Bol.com?), maar ook dat gebruikers van de database zo min mogelijk last hebben van elkaar.

### **Durability**

De effecten van een eenmaal gecommitte transactie mogen niet verloren gaan, ook niet als er een hard- of softwarestoring optreedt.

We gaan in deze introductie niet veel verder in op transactiemanagement en recovery: indien je er behoefte aan hebt zijn er meer dan genoeg bronnen te vinden met de hier gegeven trefwoorden.

## 6.2 Proeftoets

De proeftoets wordt in het college uitgewerkt.

## 6.3 Opdrachten

De opdrachten bij dit college worden op de elo gepubliceerd.

## 6.4 Verder lezen

- Een basiscursus MariaDB  
<https://www.tutorialspoint.com/mariadb/index.htm>
- Behandelt de syntax van SQL heel diepgaand.  
Het SQL leerboek, Rick van der Lans  
2012 | ISBN 9789039526552 | 7e druk
- Een overzicht en beschrijving van alle smaken NoSQL-databases  
<http://www.christof-strauch.de/nosql dbs.pdf>
- Als je een voorkeur hebt voor een boek kun je via de databanken in de Widesheimmediatheek allerlei Safari-boeken online raadplegen.

## 7 Bibliografie

- Codd, E. (1970). A relational model of data for large shared databanks. *Communications of the ACM*, 13(6), 377 - 387.
- Date, C., & Darwen, H. (2000). *Foundation for Future Database Systems: The Third Manifesto* (second ed.). Boston: Addison-Wesley.
- Halpin, T. (2001). *Information modeling and relational databases - From conceptual analysis to logical design*. San Francisco: Morgan Kaufmann Publishers.
- ISO/IEC. (2016). ISO/IEC 9075-2 Information technology — Database languages - SQL - Part 2: Foundation (SQL/Foundation). Geneve, Zwitserland: ISO/IEC.
- Lefred. (2018). *Top 10 reasons for NoSQL with MySQL*. Opgehaald van Lefred's blog: tribulations of a MySQL Evangelist: <https://lefred.be/content/top-10-reasons-for-nosql-with-mysql/>
- MariaDB. (2018). *Create User*. Opgehaald van MariaDB Knowledge Base: <https://mariadb.com/kb/en/library/create-user/>
- MariaDB(a). (2018). *Choosing the right storage engine*. Opgehaald van MariaDB Knowledge Base: <https://mariadb.com/kb/en/library/choosing-the-right-storage-engine/>
- MariaDB(c). (2018). *System tables*. Opgehaald van MariaDB Knowledge Base: <https://mariadb.com/kb/en/library/system-tables/>
- MariaDB(d). (2018). *Account Management SQL Commands*. Opgehaald van MariaDB Knowledge Base: <https://mariadb.com/kb/en/library/account-management-sql-commands/>
- MariaDB(e). (2018). *Create procedure*. Opgehaald van MariaDB Knowledge Base: <https://mariadb.com/kb/en/library/create-procedure/>
- Microsoft. (2018). *Index table pattern*. Opgehaald van Microsoft azure: <https://docs.microsoft.com/en-us/azure/architecture/patterns/index-table>
- MySQL. (2018). *Chapter 8 Optimization*. Opgehaald van MySQL 8.0 Reference manual: <https://dev.mysql.com/doc/refman/8.0/en/optimization.html>
- NIST. (2018). *Role Based Access Control*. Opgehaald van NIST - CSRC: <https://csrc.nist.gov/projects/role-based-access-control>
- NoSQL-database.org. (2018). *List of nosql databases*. Opgehaald van N\*SQL: <http://nosql-database.org/>
- Percona. (2018). *MySQL vs MongoDB - Choosing right technology for your application*. Opgehaald van Youtube: [https://www.youtube.com/watch?time\\_continue=4&v=6OEEvMnEFWY](https://www.youtube.com/watch?time_continue=4&v=6OEEvMnEFWY)
- Solid-IT. (2018). *Knowledge Base of Relational and NoSQL Database Management Systems*. Opgehaald van DB-engines: <https://db-engines.com/en/>
- Tsichritzis, D., & Lochovsky, F. (1982). *Data models*. Englewood Cliffs: Prentice-Hall, Inc.
- Tutorialspoint. (2018). *Learn MongoDB*. Opgehaald van Tutorialspoint: <https://www.tutorialspoint.com/mongodb/index.htm>
- Wiegerink, L., Bijpost, J., & de Groot, J. (2018). *Relationele databases en SQL*. Amsterdam: Boom.