Relationele databases en SQL: samenvattingen

Hoofdstuk 1

Paragraaf 1.1
Het *relationele model* is de theorie over gegevensverzamelingen bestaande uit tabellen, de wijze waarop tabellen onderling samenhangen en de regels waaraan zij moeten voldoen.

Paragraaf 1.2
Een relationele database bestaat uit aan elkaar gerelateerde *tabellen*. Een tabel is opgebouwd uit een verzameling *kolomnamen* en een verzameling *rijen*. Elke rij bestaat uit *cellen*, één per kolomnaam. Een cel bevat één kolomwaarde of is leeg. Een *kolom* wordt gevormd door alle cellen bij één kolomnaam.
Een relationele tabelstructuur mag slechts enkelvoudige celwaarden bevatten, bijvoorbeeld een getal, tekst (een rij tekens), een datum, een foto of een videofragment. Het criterium voor 'enkelvoudig' is dat het rdbms zich niet hoeft bezig te houden met de interne structuur.
Een kolom waarvan de kolomwaarden subtabellen zijn, heet een *herhalende groep*. Subtabellen voldoen niet aan de enkelvoudigheidseis. Een bezwaar ervan is dat de gegevenstaal (voor bewerkingen zoals toevoegen of verwijderen van gegevens) veel ingewikkelder zou worden. Een ander bezwaar ervan is asymmetrie: gelijkwaardige gegevens worden op een niet-gelijkwaardige manier behandeld.

Redundantie is het vóórkomen van gegevens die reconstrueerbaar zijn uit andere gegevens. Redundantie is eigen aan bepaalde structuren. Bij een redundante structuur kan het voorkomen dat gegevens onderling niet meer kloppen: *inconsistentie*. Tegen *gecontroleerde redundantie*, zoals bij een cel waarvan de inhoud automatisch wordt berekend uit andere celinhouden, bestaat geen bezwaar.

Paragraaf 1.3
Normalisatie omvat het elimineren van *herhalende groepen* en van structuren die redundantie toelaten, beide door een vorm van *tabelsplitsing* (*decompositie*).
Standaardisatie van gegevens, bijvoorbeeld om een uniforme schrijfwijze af te dwingen, kan een reden zijn om een extra tabel aan de database toe te voegen. De uiteindelijke structuur voldoet aan het principe 'elk entiteittype zijn eigen tabel'. Bij databaseontwerp wordt daar veelal direct naartoe gewerkt. Een niet-genormaliseerde database is meestal het gevolg van een ontwerpfout. Normalisatie kan dan dienen als correctiemethode.

De *populatie* van een database bestaat uit de verzameling van alle rijen van alle tabellen. De verzameling mogelijke populaties wordt ingeperkt door *beperkingsregels* (*constraints*).
Met de tekentechniek van het *strokendiagram* kan de *structuur* van een database (met de opbouw van tabellen uit hun kolommen en met de onderlinge verwijzingen) worden weergegeven.

Wanneer we een tabel opvatten als *relatie*, is deze een *verzameling* rijen. Eventuele gelijke rijen worden dan geïdentificeerd en er is geen volgorde. De term *relationele database* is ontleend aan tabellen in de zin van relaties.

Paragraaf 1.4
Syntaxis omvat de vormaspecten van taal, *semantiek* de betekenisaspecten. Bij *gegevens* gaat het om syntaxis en over zaken als opslag en manipulatie. Bij *informatie* doelen we op de betekenis van gegevens; informatie is dus een semantisch begrip.
Betekenissen van tabelgegevens kunnen worden uitgedrukt door zinnen. Deze zinnen drukken *informatie* uit. Sterker nog: elke zin drukt een *feit* uit, wat wil zeggen dat geclaimd wordt dat de zin een ware bewering uitdrukt. Zinnen met een zo klein mogelijke informatie-inhoud worden *atomaire zinnen* genoemd. Een atomaire zin drukt een *atomair feit* uit.

Hoofdstuk 2

Introductie
Een relationele database kent behalve een structuurdefinitie ook *regels*. Er zijn twee soorten regels:
– *beperkingsregels*: deze geven, in de vorm van een gebod of een verbod, aan wat allemaal mag of moet worden ingevuld als populatie
– *gedragsregels*: deze zeggen welke acties onder bepaalde voorwaarden moeten worden ondernomen, ofwel ze houden een actie juist tegen.

Paragraaf 2.1
De belangrijkste typen beperkingsregels zijn de *standaard beperkingsregels*, die in elke relationele database voorkomen.

Standaard beperkingsregels

verplichte-waarderegel voor kolom	Elke rij in die kolom moet een waarde bevatten. Een niet-ingevulde waarde heet een null. Deze regel zegt dus dat nulls niet zijn toegestaan.
uniciteitsregel voor kolom	Elke waarde in die kolom mag maar één keer voorkomen.
uniciteitsregel voor kolomcombinatie	Elke waardencombinatie in die kolomcombinatie mag maar één keer voorkomen, terwijl deze beperking niet geldt voor de afzonderlijke kolommen of 'smallere' kolomcombinaties.
kandidaatsleutel	Zo zuinig mogelijk gekozen supersleutel, in de zin dat na het verwijderen van een kolom niet meer aan de supersleuteleis wordt voldaan. Elke tabel heeft minstens één kandidaatsleutel.
primaire sleutel	Kandidaatsleutel die is gekozen om verwijzingen op te baseren.
alternatieve sleutel	Kandidaatsleutel die niet de primaire sleutel is.
referentiële-integriteitsregel	Als een kolom (of kolomcombinatie) een verwijssleutel vormt, moet elke waarde (respectievelijk waardencombinatie) voorkomen als primaire-sleutelwaarde in de doeltabel. Volgens deze regel zijn 'loze verwijzingen' niet toegestaan.
multipliciteitsregel	Regel die, voor een ouder-kind-tabelcombinatie, een minimum en/of een maximum aangeeft voor: – het aantal kindrijen bij één ouderrij – het aantal ouderrijen bij één kindrij. Deze aantallen worden grafisch genoteerd in een multipliciteitendiagram.

Buiten deze standaard beperkingsregels zijn er de *bijzondere beperkingsregels*, waarvoor geen standaardnotatie bestaat en die daarom in tekst aan een structuurdiagram worden toegevoegd.

Paragraaf 2.2
Gedragsregels hebben meestal een specifiek karakter. Er is één standaardtype, dat voor elke verwijzing (expliciet of impliciet) moet worden gespecificeerd: de *refererende-actieregels*.

Refererende-actieregels

deleteregels	Deze specificeren voor de kindtabel wat er met een rij moet gebeuren wanneer wordt gepoogd de bijbehorende ouderrij te verwijderen. Er zijn drie mogelijkheden.	
	restricted delete	verwijderen wordt tegengehouden zolang er nog een kindrij bestaat
	cascading delete	de poging heeft tot gevolg dat wordt gepoogd ook alle kindrijen te verwijderen
	nullifying delete	de poging heeft tot gevolg dat wordt gepoogd de verwijzende kolomwaarden null te maken
updateregels	Deze specificeren wat er gebeurt bij een poging tot updaten van een primaire-sleutelwaarde. Er zijn drie mogelijkheden.	
	restricted update	update wordt tegengehouden zolang er nog een of meer kindrijen bestaan met een verwijzing naar die primaire-sleutelwaarde
	cascading update	de poging heeft tot gevolg dat wordt gepoogd ook alle verwijzende kolomwaarden op dezelfde manier te updaten
	nullifying update	de poging heeft tot gevolg dat wordt gepoogd de verwijzende kolomwaarden null te maken

Gedragsregels zijn nauw verbonden met *transacties*. Een transactie is een reeks databaseacties tussen twee *commitmomenten* (momenten waarop alle wijzigingen definitief worden gemaakt). Een transactie wordt ofwel in zijn geheel, ofwel niet uitgevoerd.

Paragraaf 2.3
Een *uniciteitsregel* over een kolom of kolomcombinatie houdt een verbod in op het meervoudig voorkomen van een kolomwaarde respectievelijk waardencombinatie. De conventie is om alleen de meest strenge regels te benoemen. Bij grafische weergave door een uniciteitspijl betekent dit dat de pijl zo smal mogelijk getekend wordt. Hierdoor is het onmogelijk dat een uniciteitspijl helemaal over een andere, smallere, uniciteitspijl heen ligt.

Over drie kolommen zijn zeven uniciteitsregels mogelijk: drie over één kolom, drie over twee kolommen en één over alle drie de kolommen. Deze kunnen op allerlei manieren worden gecombineerd. Over vier kolommen is het aantal mogelijke regels veel groter.

Paragraaf 2.4
Een verwijssleutel is meer dan alleen een kolom of kolomcombinatie; het is een verbodsregel. Het verbod in kwestie is een verbod op loze verwijzingen: de *referentiële-integriteitsregel*. Wanneer verwezen wordt naar een samengestelde primaire sleutel, is de verwijssleutel eveneens samengesteld.
Een bijzonder geval van een verwijssleutel is een *recursieve verwijzing*: een verwijzing naar rijen van de eigen tabel.
Een verwijssleutel is gebaseerd op sleutelwaardengelijkheid: die van de verwijssleutelwaarde zelf en die van de primaire sleutel waarnaar wordt verwezen. Een verwijzing kan echter ook op een andere manier tot stand komen. Een voorbeeld van zo'n niet-sleutelverwijzing is een *intervalverwijzing*.

Hoofdstuk 3

Paragraaf 3.2
Er zijn SQL-subtalen voor uiteenlopende taken:
- DDL (*Data Definition Language*) voor het definiëren van structuren en het vastleggen van regels
- DML (*Data Manipulation Language*) voor het uitvoeren van operaties voor het opvragen of wijzigen van gegevens
- DAL (*Data Authorization Language*) voor de autorisatie (het regelen van de gebruikerstoegang tot de gegevens)
- DCL (*Data Control Language*) voor beheertaken met betrekking tot de prestaties (performance).

Elk rdbms heeft zijn eigen SQL-dialect. Het ANSI en het ISO hebben met enig succes gepoogd SQL te standaardiseren. Een belangrijke ANSI/ISO-standaarden is SQL2 (ook SQL92 genaamd). Een meer recente is SQL:2008.

Paragraaf 3.3
De Boekverkenner biedt onder meer de volgende faciliteiten: zoeken in de tekst, een Voorbeeldnavigator, bijhouden van navigatiehistorie, installeren of verwijderen van voorbeelddatabases en inloggen op een voorbeelddatabase. Statements in de boektekst kunnen worden aangeklikt en uitgevoerd. Hiertoe werkt de Boekverkenner samen met de Interactive Query Utility (IQU, een SQL-querytool bij het Firebird-rdbms).

Paragraaf 3.4
De SQL-opdracht om gegevens op te vragen, luidt: `select`. Dit is één van de opdrachten van de subtaal DML (*data manipulation language*). De `from`-clausule (zinsdeel) bevat de brontabel.
De SQL-interpreter (het onderdeel van het rdbms dat binnenkomende SQL-statements verwerkt) kijkt niet naar de opmaak. Vanuit het oogpunt van de programmeerstijl is deze echter van groot belang.

Woorden die tot de vaste taalschat van SQL behoren, heten *gereserveerde woorden* (keywords). Voor door de programmeur zelf gekozen woorden gelden 'harde' syntaxregels (het moeten 'identifiers' zijn). Daarnaast zijn 'zachte' regels op het gebied van programmeerstijl van belang.

Een *null* wordt in een Firebird-resultaattabel weergegeven als <NULL>. Een *lege string* (string zonder tekens) wordt weergegeven door 'niets'.
Door een- of meerregelige commentaarcode kan SQL-code voor de menselijke lezer worden verduidelijkt.

Paragraaf 3.5
Via de `select`-clausule kan een *projectie* worden uitgevoerd van de brontabel op een deelverzameling van de kolommen. Een projectie (in gegeneraliseerde zin) mag ook constante of berekende kolommen bevatten.
Via de `where`-clausule kan een conditie worden meegegeven, waardoor een *selectie* wordt uitgevoerd op een deelverzameling van de rijen.

Met een `order by`-clausule kan het eindresultaat worden geordend. Dit is geen relationele operatie.
De *conceptuele verwerkingsvolgorde* is: brontabel (`from`), selectieconditie (`where`), projectie (`select`), ordening (`order by`).

Paragraaf 3.6
De `select`-lijst en de `where`-conditie mogen samengestelde expressies bevatten, met operatoren en functies. Een *operator* levert een resultaatwaarde door bewerking op een of meer *operanden*. Een functie heeft een functienaam en werkt op nul of meer *argumenten*.

Paragraaf 3.7
De *join* is een bewerking op twee tabellen, die resulteert in 'zinvolle' combinaties van rijen van de ene tabel met rijen van de andere tabel. De hier behandelde join is op te vatten als een *verbrede tabel*: een kindtabel verbreed met bijbehorende rijen van een oudertabel. Zo'n bijbehorende rij wordt gevonden via een verwijzing.

Paragraaf 3.8
De SQL-opdrachten om een nieuwe rij toe te voegen, rijen te verwijderen en rijen te wijzigen, luiden respectievelijk: `insert`, `delete` en `update`. Deze behoren net als `select` tot de DML.
SQL-statements mogen met elkaar worden gecombineerd tot een SQL-programma, *script* genaamd. Voor elke voorbeelddatabase zijn twee scripts beschikbaar: een createscript om de (lege) tabellen aan te maken en een insertscript om een eventuele oude rijenpopulatie te verwijderen en een nieuwe aan te maken.
De SQL-opdrachten `commit` en `rollback` beëindigen een *transactie*. Een succesvolle `commit` maakt alle wijzigingen definitief, een `rollback` maakt alles ongedaan. Beide behoren tot de SQL-subtaal *transaction control language* (TCL).

Paragraaf 3.9
De *data definition language (*DDL*)* is de subtaal van SQL waarmee de structuur van een database kan worden gedefinieerd en onderhouden. De DDL omvat onder meer `create`-opdrachten, waarmee databaseobjecten (waaronder tabellen) worden gecreëerd.
Een *gebruiker* is een combinatie van een gebruikersnaam en een wachtwoord.
De opdracht om een tabel aan te maken luidt: `create table`. Een volledig `create table`-statement bevat onder meer de tabelnaam, de kolomnamen, voor elke kolom een datatype en voor verplichte kolommen een `not null`-aanduiding. Een primaire sleutel wordt gedefinieerd via een `primary key`-clausule. Een verwijssleutel wordt gedefinieerd via een `foreign key`-clausule, eventueel met aanduiding voor refererende acties (zoals `on update cascade` of `on delete cascade`). Constraints zijn de technische realisaties van databaseregels (waaronder primaire sleutels en verwijssleutels).

Paragraaf 3.10
De verzameling van alle systeemfuncties, de diensten die het systeem de gebruiker heeft te bieden, heet de *functionaliteit* van het systeem.
De grafische applicatie Reijnders' Toetjesboek illustreert drie elementaire systeemfuncties: bladeren, zoeken en onderhoud. Ouder-kind-relaties tussen databasetabellen zien we op verschillende manieren terug in de applicatie: via *master-detailvensters* en via keuzelijsten (*lookups*).
Bladervensters hebben een voorgeprogrammeerde *onderliggende* `select`-*query* (of meerdere, bij een master-detailvenster). Een zoekvenster stelt dynamisch een `select`-query samen uit de zoekcondities die de gebruiker heeft ingevuld.
Onderhoud is bedoeld om de database-inhoud te wijzigen: rijen toevoegen (`insert`), verwijderen (`delete`) of veranderen (`update`). Om de aanwezige inhoud te tonen, moet ook uit de database worden gelezen (`select`). Onderhoudsvensters zijn meestal eenvoudig, omdat ze nauw aansluiten bij de tabelstructuur. Ook zij hebben voorgeprogrammeerde onderliggende `select`-query's om de 'huidige' tabelinhouden op te halen.

Hoofdstuk 4

Paragraaf 4.1
Een *null* is een speciale, typeloze waarde, die voor de gebruiker de betekenis heeft van 'geen waarde ingevuld'. SQL heeft hiervoor de constante `null`. Een `null`-operand bij een optelling geeft een `null`-uitkomst. Bij concatenatie is dit dialectafhankelijk.

Een *lege string* is een string zonder tekens, dus van lengte 0. Een lege string is conceptueel iets anders dan een null. Sommige SQL-dialecten vatten een lege string op als null. Concatenatie van een string met een null kan een resultaat opleveren dat verschilt per SQL-dialect.
Een null kan erop duiden dat op die plek een waarde (nog) onbekend is of niet van toepassing. Dit is vanuit de optiek van de gebruiker; null is immers zelf een databasewaarde.

Paragraaf 4.2
Het historische relationele model volgens Codd is, voor zover het over sleutels gaat, gebaseerd op primaire sleutels. Modernere versies stellen vaak het begrip kandidaatsleutel centraal.
Codd eiste voor brede primaire sleutels dat elke sleutelkolom not-null is. Deze eis heet *Codd-relationaliteit*. Men vindt haar terug in vrijwel alle commerciële SQL-rdbms'en.

Paragraaf 4.3
Logische expressies kunnen worden samengesteld met, in volgorde van prioriteit, de logische operatoren `not`, `and` en `or`.
In de *tweewaardige logica* hebben die expressies één van de waarden *true* of *false*. De regels om de logische uitkomst van een logische expressie te berekenen, liggen vast in *waarheidstabellen*.
De *driewaardige logica* werkt met een derde logische waarde: *unknown*. Deze is noodzakelijk vanwege nulls als operand in vergelijkingsexpressies. De waarheidstabellen van de driewaardige logica vormen een uitbreiding van die van de tweewaardige logica.
Wanneer een selectieconditie van een SQL-statement de waarde *unknown* heeft, heeft dit hetzelfde effect als *false*: de betreffende rij draagt niet bij aan de resultaattabel.

Hoofdstuk 5

Paragraaf 5.1
Normaliseren houdt in: elimineren van herhalende groepen en elimineren van redundantie. Een structuur zonder herhalende groepen staat per definitie in de eerste normaalvorm (1NV) en heet *genormaliseerd*.

Paragraaf 5.2
Een kolom B heet *functioneel afhankelijk* van een kolom of kolomcombinatie A in dezelfde tabel wanneer de regel geldt dat bij elke A-waarde steeds dezelfde B-waarde hoort. We zeggen: A *determineert* B. A heet de *determinant*, B de *gedetermineerde*. We zijn alleen geïnteresseerd in niet-triviale gevallen, dat wil zeggen dat B niet gelijk is aan A of een deelverzameling is van A.

Wanneer A een kandidaatsleutel is, is elke B functioneel afhankelijk van A. Dit is het normale en 'gewenste' geval van functionele afhankelijkheid.

Wanneer één bepaalde waarde A daadwerkelijk vaker voorkomt, zijn de meervoudige vermeldingen van de corresponderende B-waarden redundant. Deze (meestal ongewenste) situatie kan zich voordoen wanneer voor A geen uniciteitsregel geldt.

Paragraaf 5.3
De *tweede normaalvorm* (*2NV*) eist 1NV en verbiedt *partiële sleutelafhankelijkheid*: een niet-sleutelkolom die functioneel afhankelijk is van een echt deel van een kandidaatsleutel.

Een structuur die 2NV overtreedt kan redundante gegevens bevatten. De oplossing is: een structuurwijziging, waarbij de combinaties van determinant (A) en gedetermineerde (B) in een eigen tabel worden geplaatst waarvan A de primaire sleutel wordt en waarin eenmalig ('single point of definition') bij elke A-waarde de B-waarde wordt vermeld. De B-kolom verdwijnt uit de oorspronkelijke tabel, de A-waarden daarin worden verwijzingen naar de nieuwe tabel.

Paragraaf 5.4
De *derde normaalvorm* (*3NV*) eist 2NV en verbiedt *transitieve sleutelafhankelijkheid*: een niet-sleutelkolom die indirect (via een andere niet-sleutelkolom of -combinatie) functioneel afhankelijk is van een kandidaatsleutel.
Ook een structuur die 3NV overtreedt kan redundante gegevens bevatten. De oplossing is: een structuurwijziging analoog aan die voor 2NV.

Paragraaf 5.5
De *Boyce-Codd-normaalvorm* (BCNV) is nog wat sterker dan 3NV. Een tabel voldoet aan BCNV als hij voldoet aan 1NV en geen functionele afhankelijkheid bevat met een niet-unieke determinant.

Een kritisch voorbeeld (3NV en niet BCNV) leert ons dat het uitbannen van een *intra*tabelregel (een redundantie veroorzakende functionele afhankelijkheid) een *inter*tabelregel kan introduceren (een joinuniciteitsregel). De vraag is gerechtvaardigd wat we daarmee zijn opgeschoten. BCNV is niet beter dan 3NV.

We moeten erop bedacht zijn dat de exacte definities van 2NV en van 3NV wel eens kunnen verschillen. In oudere literatuur bijvoorbeeld kunnen ze zijn gebaseerd op alleen primaire sleutels.

De Boyce-Codd-normaalvorm komt, voor kritische gevallen (wel 3NV en niet BCNV), neer op het verruilen van ongewenste intratabelkenmerken voor ongewenste intertabelkenmerken.

Paragraaf 5.6
De 1NV-eis (geen herhalende groepen) voor de hogere normaalvormen is theoretisch gezien niet vanzelfsprekend. Structuren die redundantie toestaan (geen 3NV) kunnen soms nuttig zijn, met name als de performance erdoor verbeterd wordt.

Hoofdstuk 6

Paragraaf 6.1
Een `select`-statement is een relationele operator, werkend op een tabel als operand (de brontabel) en met een tabel als resultaatwaarde.
Een eenvoudige vorm ervan is de projectie op een deelverzameling van de kolommen. Bij een gegeneraliseerde vorm van de projectie mag de `select`-lijst ook constanten of berekende expressies bevatten.
Elementen uit de `select`-lijst kunnen een kolomalias krijgen, die is op te vatten als een gewone kolomnaam van de resultaattabel.
Door de `select`-lijst vooraf te laten gaan door `distinct`, worden dubbele rijen in de presentatie van de resultaattabel onderdrukt.

Paragraaf 6.2
Gegevens in één kolom zijn van een bepaald datatype. De SQL-standaard schrijft onder meer voor: `integer` (gehele getallen), `numeric` (algemenere numerieke waarden), `char` (tekst van vaste lengte), `varchar` (tekst van variabele lengte) en `date` (kalenderdata).
Bij tekst (alfanumerieke tekenrijen) kan een *character set* zijn gedefinieerd, die vastlegt hoe de intern-geheugenrepresentatie van tekens dient te worden vertaald naar een bepaalde verschijningsvorm van het teken, en omgekeerd. Veel character sets hebben de ASCII-set als deelverzameling (de eerste 128 tekens).
Via *typecasting* kan soms een waarde van het ene datatype in een waarde van het andere datatype worden geconverteerd.

Paragraaf 6.3
Een *operator* is een teken of samenstel van tekens waarmee een bewerking wordt gedefinieerd. Een operator werkt op een of meer *operanden*. De meeste operatoren zijn binair (werkend op twee operanden). Afhankelijk van het datatype van de operanden onderscheiden we numerieke operatoren, alfanumerieke operatoren, enzovoort.
Soms is expliciete typecasting nodig. Soms gebeurt dit automatisch.

Paragraaf 6.4
Een *functie* retourneert, net als een operator, een resultaatwaarde op de plaats waar zij wordt aangeroepen. Een functie heeft een naam en werkt op nul of meer *argumenten*, die in de aanroep tussen haakjes worden meegegeven. Afhankelijk van het datatype van de argumenten (soms van de geretourneerde waarde) spreekt men van numerieke functies, datumfuncties, enzovoort. Vergelijkbaar met de operatoren voor typecasting zijn de conversiefuncties, in het bijzonder die voor conversie van tekst naar datum/tijd (in een specifiek formaat) of omgekeerd.
Een belangrijke mogelijkheid is zelf functies te definiëren: *user defined functions* (*udf*'s).

Paragraaf 6.5
Door een `select`-statement uit te breiden met een `where`-clausule met selectieconditie, beperken we de resultaattabel tot de rijen die aan de conditie voldoen: een selectie.
Een selectieconditie is een logische expressie van de driewaardige logica.
Belangrijke operatoren om logische expressies mee te vormen zijn de vergelijkingsoperatoren (=, <>, >, <, >=, <=), de ternaire operator `between ... and`, de tekstvergelijkingsoperator `like`, de 'is element van'-operator `in`. Van bijzonder belang zijn de operatoren `is null` en `is not null` voor het selecteren op lege of juist niet-lege cellen
Om samengestelde logische expressies te vormen dienen de logische operatoren `and`, `or` en `not`. Hierbij is het van belang te letten op de prioriteit van de operatoren en zo nodig haakjes te gebruiken.

Paragraaf 6.6
Door een `select`-statement uit te breiden met een `order by`-clausule, kan een ordening (klimmend of dalend) van de resultaattabel worden afgedwongen op één van de elementen van de `select`-lijst. Ook kan *verfijnd* worden geordend op een combinatie van elementen. Het gaat hierbij om *presentatie*; vanuit relationeel oogpunt is elke tabel ongeordend.

Paragraaf 6. 7
De verzamelingsoperator `union` werkt op twee tabeloperanden, waarvan de kolommen twee aan twee van vergelijkbaar datatype moeten zijn (bijvoorbeeld: beide alfanumeriek). De operator `union` komt overeen met de verenigingsoperator uit de verzamelingenleer. De noodzaak `union` te gebruiken, kan duiden op een slecht ontworpen database.
De SQL-standaard kent ook de verzamelingsoperatoren `intersect` en `except`, die overeenkomen met de doorsnede- en verschiloperator uit de verzamelingenleer. Expressies met `intersect` en `except` kunnen vrijwel altijd worden herschreven door één `select`-expressie met een geschikte selectieconditie.

Hoofdstuk 7

Paragraaf 7.1
De inner join over een verwijssleutel kan worden gezien als een operatie op de kindtabel, waarvan de rijen worden verbreed met corresponderende waarden uit de oudertabel. Bij een optionele verwijssleutel verdwijnen de rijen waarvoor de verwijssleutelwaarde null is.
Joinen is een vorm van denormaliseren: de resultaattabel kan redundante gegevens bevatten. Omdat het hier niet gaat om gegevensopslag, is dit geen bezwaar.
De inner join kan in SQL worden gerealiseerd door uit de producttabel (cartesisch product) de rijen te selecteren met corresponderende sleutelwaarden en het resultaat vervolgens te projecteren op alle kolommen behalve de primaire sleutel van de oudertabel.

Paragraaf 7.2
Bij gegeven verwijssleutel kan de left outer join van de kindtabel met de oudertabel (in die volgorde) worden gezien als een inner join, uitgebreid met de (met nulls verbrede) rijen van de kindtabel waarvoor de verwijssleutelwaarde null is.
In elk SQL-dialect kan de left outer join worden gerealiseerd als een `union`-expressie: de inner join verenigd met de extra rijen.

Paragraaf 7.3
In de SQL-standaard en veel SQL-dialecten kunnen de inner join en de left outer join worden gerealiseerd met de joinoperatoren `inner join` (die mag worden afgekort tot `join`) en `left outer join`. De joinconditie (meestal: sleutelgelijkheid) wordt ondergebracht in een `on`-clausule van de joinexpressie.

Paragraaf 7.4
Joinen over een brede sleutel impliceert een samengestelde joinconditie.

Paragraaf 7.5
Inner joins en outer joins kunnen worden samengesteld tot complexe joinexpressies. Ongeacht of deze worden gerealiseerd middels een cartesisch product of met joinoperatoren, vereist een goede programmeerstijl dat de volgordes van tabellen en kolommen worden aangepast aan het navigatiepad.

Paragraaf 7.6
Via een recursieve verwijzing kan een tabel worden gejoind met zichzelf. Het resultaat heet een (inner of outer) autojoin. Een autojoin kan worden gezien als een gewone join van twee virtuele exemplaren van de tabel, elk met een eigen naam (alias). Realisatie van een autojoin in SQL gaat, door het gebruik van tabelaliassen, dan ook precies als bij gewone joins.

Paragraaf 7.7
Een join hoeft niet gebaseerd te zijn op een verwijssleutel en dus op sleutelgelijkheid. Er kan ook worden gejoind over een niet-sleutelverwijzing, zoals een intervalverwijzing.

Paragraaf 7.8
De right outer join correspondeert met de left outer join, zij het dat nu van de rechter joinoperand geen enkele rij verloren gaat. Gebruik van de right outer join kan duiden op een niet-consequente programmeerstijl. Uitgaan van de juiste starttabel en aanpassing van alle volgordes aan het navigatiepad leidt altijd tot de left-variant.
De full outer join kan worden gezien als de vereniging (union) van een left outer join en een right outer join. Full-outer-joinoverzichten zijn hybride overzichten, in de zin dat ze niet over één entiteittype gaan. Het nut ervan is daarom twijfelachtig.

Hoofdstuk 8

Paragraaf 8.1
Voor het verkrijgen van statistische informatie is een nieuw concept nodig: *groeperen*. Ook is een nieuw soort functies nodig: statistische functies. Deze werken op verzamelingen van rijen of waarden.

Paragraaf 8.2
Eenvoudige voorbeelden van statistische informatie krijgt men door alle rijen van een tabel (die het resultaat mag zijn van een selectie, met `where`) als één groep op te vatten en hiervan het aantal rijen te tellen (`count`), de waarden in een numerieke kolom op tellen (`sum`) of te middelen (`avg`) of van een kolom het maximum (`max`) of minimum (`min`) te nemen.
Statistische functies forceren een groepsbenadering, die afwijkt van de normale rij-voor-rij verwerking. In dit verband dient men op te passen voor 'scheve query's'.

Paragraaf 8.3
Met de clausule `group by` kan een tabel (die ook weer het resultaat mag zijn van een selectie, met `where`) worden opgedeeld in groepen van rijen, met gemeenschappelijke kolomwaarde (het groeperingskenmerk). Een `select`-query met `group by` geeft een resultaatrij met één rij per groep. Zo'n rij bevat doorgaans het groeperingskenmerk en een of meer statistische waarden per groep.
Er kan *verfijnd* worden gegroepeerd op een combinatie van kolomwaarden.
Een extra `having`-clausule fungeert als selectieoperator op groepen die voldoen aan een opgegeven conditie. De conditie is gesteld in termen van groepskenmerken (groeperingskenmerken en/of statistische waarden).
Het is mogelijk te groeperen op een berekende expressie.
Groeperen is conceptueel gezien een vorm van denormaliseren: het resultaat ervan (na de `group by`-operatie en vóór de `select`) is een tabel met een herhalende groep.

Paragraaf 8.4
Bij het groeperen van een join is het navigatiepad belangrijk. De navigatie start, als altijd, bij de tabel die de objecten (rijen) bevat waarover informatie wordt gevraagd.
Wanneer statistische kenmerken van een kindtabel worden verlangd, moet worden genavigeerd van ouder (starttabel) naar kindtabel. Dat is in de richting één-veel. Bij één primaire-sleutelwaarde van de oudertabel (startpunt van de navigatie) kunnen meerdere kindrijen horen. Gegevens van de kindtabel worden daarbij statistisch verdicht tot kenmerken van een ouderrij.

Doordat de SQL-interpreter het feit negeert dat alle kolommen functioneel afhankelijk zijn van de primaire sleutel, is het nodig verfijnd te groeperen op een extra kolomwaarde van de oudertabel (naast de primaire sleutel, die het eigenlijke groeperingskenmerk vormt), wanneer deze voorkomt in de `select`-lijst.

Het groeperen van een left outer join (oudertabel linker operand, kindtabel rechter operand) vraagt extra aandacht wanneer ook statistische kenmerken worden gevraagd van 'ouders zonder kind'. In dit verband is van belang dat `count(*)`, in combinatie met `group by`, nooit 0 is.

Paragraaf 5
De resultaattabel van een statistische query kan opnieuw worden gegroepeerd: genest groeperen.
Minimaxproblemen zijn problemen waarbij een tussenresultaattabel met statistische waarden tot één statistische waarde moet worden herleid, bijvoorbeeld: het minimum van een aantal maxima, of het gemiddelde van een aantal totalen. Minimaxproblemen kunnen onder andere worden opgelost door statistische functies te nesten.

Paragraaf 8.6
Een conceptueel algoritme is een denkbeeldige wijze van verwerken die de menselijke lezer inzicht geeft in de betekenis (werking) van een statement. Het conceptuele algoritme voor een `select`-statement komt neer op het achtereenvolgens toepassen (voor zover van toepassing) van de volgende relationele operatoren op een brontabel: selectie (`where`), groeperen (`group by`), selectie van groepen (`having`), projectie (`select ... from`). Als laatste kan voor presentatiedoeleinden nog worden geordend (`order by`).
De brontabel kan een databasetabel zijn, of zelf ook het resultaat van een of meer relationele operaties (een tabelproduct of een join).

Paragraaf 8.7
Het formuleren van zinvolle statistische query's vereist dat groeperingskenmerken gestandaardiseerd zijn. Dit is een kwestie van goed databaseontwerp (een standaardisatietabel of een datatype met discrete waarden).

Hoofdstuk 9

Paragraaf 9.1
Een subselect is een `select`-expressie binnen een `select`-, `update`- of `delete`-statement. Een subselect is in principe overal toegestaan waar hij een waarde oplevert die conceptueel gezien zinvol is. De mogelijkheden zijn per dialect verschillend. Binnen een `select`-statement kan een subselect optreden binnen de `select`-, `from`-, `where`- of `having`-clausule.

Een niet-gecorreleerde subselect kan zelfstandig uitgevoerd worden en is onafhankelijk van enige rij of groep van de hoofdselecttabel. Een gecorreleerde subselect kan niet zelfstandig uitgevoerd worden en is afhankelijk van een 'bijbehorende' rij of groep van de hoofdselect.

Een subselect kan worden gezien als de oplossing van een deelprobleem en als hulpmiddel bij het stapsgewijs oplossen van een complex probleem.

Paragraaf 9.2
Bij subselects, met name bij gecorreleerde subselects, wordt door de database genavigeerd op een manier die vergelijkbaar is met joinnavigatie. Meestal is dat op basis van sleutelverwijzingen. Net als bij een join kan dat zijn over een brede sleutel.
Soms is het gebruik van subselects een alternatief voor een join. Subselects kunnen op allerlei manieren worden gecombineerd met joins.

Paragraaf 9.3
Het moment waarop een subselect (conceptueel gezien) wordt geëvalueerd, kan worden afgeleid uit het conceptuele algoritme. Voor een subselect binnen een `where`-clausule is dit per rij van de hoofdselect. Voor een subselect binnen een `having`-clausule is dit per groep van het gegroepeerde tussenresultaat.
De operator `exists` werkt op een subselect als operand en retourneert een logische waarde: *true* wanneer de subselect een niet-lege tabel oplevert, en anders *false*. De subselect is vrijwel altijd gecorreleerd: het gaat om het wel of niet bestaan van een 'bijbehorende' rij.
Een gecorreleerde subselect met (`not`) `exists` is vaak equivalent met een niet-gecorreleerde subselect met (`not`) `in`.

Paragraaf 9.4
Een subselect mag een subselect bevatten: een geneste subselect.
Sommige problemen met 'alle' kunnen worden opgelost door geneste subselects met `not exists`. Dergelijke vraagstellingen gaan vaak uit van de 'closed world assumption', volgens welke de database-inhoud een volledige afspiegeling is van de relevante wereld.

Paragraaf 9.5
Subselects met `all` of met `any` maken soms formuleringen mogelijk die dicht bij de natuurlijke taal staan. Ze dragen echter weinig bij aan het oplossingsrepertoire, omdat er alternatieven zijn met een betere logische structuur.

Paragraaf 9.6
Views zijn `select`-query's met een naam, die in andere query's gebruikt kunnen worden als een tabel. Niet-gecorreleerde `select`-query's kunnen ook als view gedefinieerd worden. Views vormen een krachtig hulpmiddel bij het stapsgewijs oplossen van problemen.

Paragraaf 9.7
Vaak is keuze mogelijk tussen een joinaanpak en stapsgewijze verfijning. Bij de joinaanpak worden alle betrokken tabellen als het ware 'platgeslagen' in een jointabel, waaruit – al dan niet na groepering – de benodigde gegevens kunnen worden geselecteerd.
Bij stapsgewijze verfijning lossen we het probleem op in stappen, waarbij we naast SQL natuurlijke taal gebruiken voor het formuleren van deelproblemen. Deze deelproblemen leiden tot subselects.
Beide aanpakken kunnen gecombineerd voorkomen: voor het ene deelprobleem de joinaanpak, voor het andere de subselectaanpak.

Hoofdstuk 10

Paragraaf 10.1
Op grond van het type transacties dat op de database wordt uitgevoerd, onderscheiden we:
- systemen voor *Online Transaction Processing* (OLTP), voor het snel, online verwerken van een grote hoeveelheid nieuwe gegevens
- *transactionele bedrijfssystemen*: voor het bijhouden van de dagelijkse bedrijfswerkelijkheid
- *datawarehouses*: grote historiedatabases met gegevens uit verschillende bronnen, voor het opvragen van managementinformatie en het uitvoeren van analyses.

De term OLTP wordt vaak breder gebruikt en omvat dan mede de genoemde transactionele bedrijfssystemen.

Paragraaf 10.2
Wijzigingen worden definitief door een *commitmoment*. Dit valt samen met het uitvoeren van het `commit`-statement, of – bij sommige rdbms'en of SQL-omgevingen – bij een commit als neveneffect van een DDL-statement.
Het moment van starten of beëindigen van een transactie wordt bepaald door het *transactiemodel*. Bij het *impliciete model* wordt een transactie automatisch gestart na het beëindigen van de voorgaande.

Paragraaf 10.3
Bij het wijzigen van een database-inhoud moet het handhaven van database-integriteit gewaarborgd zijn. Soms is het daarbij wenselijk een beperkingsregel tijdens een transactie tijdelijk uit te schakelen.
Een belangrijk mechanisme om referentiële integriteit te handhaven, wordt gevormd door de delete- en updateregels.

Paragrafen 10.4, 10.5 en 10.6
Het `insert`-statement, voor het invoeren van nieuwe rijen, kent twee varianten: een met een `values`-lijst voor het invoeren van één rij en een met een `select`-clausule voor het invoeren van een verzameling rijen. In de variant met de `values`-lijst is het specificeren van een kolommenlijst verplicht, tenzij de `values`-lijst, in lengte en volgorde, correspondeert met de kolomdefinitie van het `create table`-statement.
Het `delete`-statement en het `update`-statement, voor achtereenvolgens het verwijderen en wijzigen van rijen, kennen – optioneel – een `where`-clausule met selectieconditie. Deze mag een subselect bevatten, al dan niet gecorreleerd.
Het `update`-statement bevat daarnaast een verplichte `set`-clausule met een expressie waarin de nieuwe celwaarden worden gespecificeerd. Ook deze expressie mag een al dan niet gecorreleerde subselect bevatten.
Bijzondere aandacht verdienen updates van sleutelwaarden.

Hoofdstuk 11

Paragraaf 11.2
De DDL-subtaal van SQL bevat commando's voor het definiëren (`create`) of verwijderen (`drop`) van databaseobjecten en het wijzigen van hun structuur (`alter`).
Databaseobjecten zijn er van velerlei typen. Naast tabellen zijn er *domeinen* (een soort logische gegevenstypen), *views* (opgeslagen `select`-statements met een naam), *sequences* (objecten die een volgnummer kunnen genereren), *triggers* (3GL-programmaatjes die automatisch actief worden bij bepaalde wijzigingen van de database-inhoud of pogingen tot wijziging), *stored procedures* (eveneens 3GL-programmaatjes die worden uitgevoerd bij een expliciete aanroep), *indexen* (toegangsstructuren ter verhoging van de performance), *gebruikers* (bepaald door gebruikersnaam en wachtwoord) en *rollen* ('bundeltjes' privileges).

Tabellen zijn er in twee typen: tabellen met gebruikersinformatie en tabellen met informatie over databaseobjecten (meta-informatie). De laatste vormen samen de *data dictionary* of *data dictionary*.
Een DDL-statement vormt in het algemeen geen commitmoment. In bepaalde dialecten of omgevingen (zoals IQU) kan dit wel het geval zijn.
Structuurwijzigingen worden, wisselend per SQL-dialect, soms slecht ondersteund. Voor sommige 'deltaproblemen' kan daarom een complexe omweg noodzakelijk zijn.

Paragraaf 11.3
Een database wordt gecreëerd met een `create database`-statement en verwijderd met `drop database`.
Een *sessie* is de periode dat één gebruiker vanuit één instantie van een clientapplicatie een *connectie* (verbinding) met een database onderhoudt. Een sessie beginnen (*inloggen*) gaat in Firebird/IQU met het commando `connect` (of automatisch na het creëren van de database), een sessie beëindigen (*uitloggen*) met `disconnect`.

Paragraaf 11.4
Een tabel wordt gecreëerd met een `create table`-statement. Zo'n statement bevat kolomdefinities en constraintdefinities. Constraints (onder meer voor primaire of verwijssleutels) kunnen een naam krijgen door het gereserveerde woord `constraint` gevolgd door de constraintnaam.
Het commando om een tabel te verwijderen, luidt: `drop table`. Of een poging slaagt, hangt onder meer af van referentiële integriteit.
Structuurwijziging van een tabel (inclusief naamwijziging of wijziging van constraints) kan soms worden bewerkstelligd met het commando `alter table`.

Paragraaf 11.5
Een kolomdefinitie (onderdeel van een `create table`-statement) bevat verplicht een kolomnaam en een datatype, en optioneel een `not null`- en/of `default`-specificatie. Het datatype hierin is een *extern datatype*, te onderscheiden van de door het rdbms gebruikte *interne datatypen*.

Paragraaf 11.6
Er zijn de volgende typen constraints: `primary key`-, `foreign key`-, `unique`- en `check`-constraints. Een unique-constraint legt, samen met een `not null` een alternatieve sleutel vast. Met een `check`-constraint kan een conditie worden vastgelegd die wordt gecontroleerd bij een insert of een update.

Paragraaf 11.7
Het begrip *domein* komt tegemoet aan de behoefte aan een 'single point of definition' voor kenmerken die bepaalde kolommen a priori gemeenschappelijk hebben (in het bijzonder het datatype, zoals bij een primaire sleutel en de corresponderende verwijssleutels).

Paragraaf 11.8
Een *view* is een `select`-statement dat onder een eigen naam wordt bewaard. Views mogen in `select`-statements overal worden gebruikt waar een databasetabel is toegestaan. Views kunnen worden gebruikt ter oplossing van deelproblemen van een complex probleem (als zodanig ook om syntactische beperkingen van SQL te omzeilen). Een belangrijke toepassing is het verstrekken van rechten 'op maat' aan gebruikers.

Paragraaf 11.9
Met een *sequence* kunnen automatisch volgnummers worden gegenereerd. De sequence 'onthoudt' waar hij gebleven was. In de praktijk wordt voor elke kunstmatige sleutel een sequence gedefinieerd. Deze wordt aangeroepen vanuit een `insert`-statement (soms ook bij een update van een primaire sleutel).

Hoofdstuk 12

Paragraaf 12.1
De DBA (*database administrator*) is verantwoordelijk voor het goed functioneren van het databasesysteem: gebruikersbeheer (inclusief autorisatie), performance en veiligheid. Een DBA is een 'supergebruiker' met specifieke rechten.
Databaseomgevingen verschillen soms sterk in de *database-user-structuur*. Deze bepaalt onder meer of gebruikers eigenaar kunnen zijn van een gehele database en zelf los daarvan bestaan, dan wel zelf object zijn bínnen een database. De database-user-structuur heeft vele consequenties: op het gebied van eigenaarschap, systeemrechten en identificatie van objecten.
Taken op het gebied van gebruikersbeheer zijn dialectspecifiek.

Paragraaf 12.2
Systeemprivileges zijn rechten die betrekking hebben op het mogen starten van sessies, het mogen creëren van databases of databaseobjecten en op DBA-taken.

Objectprivileges zijn rechten op specifieke databaseobjecten:
– op tabellen en views: `select`, `insert`, `delete`, `update`
– specifiek op tabellen: `references` (het recht een constraint aan te maken die naar de tabel verwijst)
– op stored procedures: `execute`.

Via het gereserveerde woord `all` kunnen `select`, `insert`, `delete`, `update` en `references` in één keer worden toegekend.
Niet alleen gebruikers kunnen rechten krijgen: ook rollen (paragraaf 4), triggers en stored procedures.
Privilegetoekenning aan de eigenaar van een object is niet aan de orde: de eigenaar heeft alle rechten bij voorbaat.

Privileges worden toegekend met: `grant` <*één of meer privileges*> `on` <*één object*> `to` <*één of meer gebruikers*>. Ze worden teruggenomen via een analoog: `revoke` ... `on` ... `from` ...-statement.

De pseudogebruiker `public` dient voor rechtentoekenning aan alle (ook toekomstige) gebruikers.
Een recht dat is ontvangen via een `grant`-statement met de clausule `with grant option`, mag aan derden worden doorgegeven.

Paragraaf 12.3
Met rechten op views is verfijnde autorisatie mogelijk, bijvoorbeeld op een tabelselectie.
Belangrijkste kenmerk van een updatable view is dat een viewrij is te herleiden tot precies één rij van een onderliggende tabel. Op een updatable view kunnen DML-opdrachten worden uitgevoerd, voor onderhoud op de onderliggende tabel. Door aan de viewdefinitie de clausule `with check option` toe te voegen, gaat de selectieclausule ook als filter voor invoercontrole werken.

Paragraaf 12.4
Een rol is een databaseobject waaraan rechten kunnen worden toegekend en dat zelf kan worden toegekend aan gebruikers of eventueel aan triggers, stored procedures of andere rollen. Als zodanig is een rol te beschouwen als een abstracte gebruiker (uitvoerder van een bedrijfsproces), met een corresponderende verzameling privileges.
De details zijn dialectafhankelijk. In Firebird zijn de privileges die aan een rol zijn verbonden, slechts beschikbaar indien men met expliciete rolvermelding inlogt.

Hoofdstuk 13

Paragraaf 13.1
In de context van meer technische onderwerpen, zoals performance, worden rijen vaak aangeduid met de wat meer fysiek georiënteerde term *records* en kolommen of kolomwaarden met *velden*.

Paragraaf 13.2
Na controle door de *parser* op syntactische correctheid, wordt een query onder handen genomen door de *optimizer*, die probeert een verwerkingsalgoritme te genereren met optimale performance (snelheid) en geheugengebruik. Dit verwerkingsalgoritme is gebaseerd op een *queryplan*. Het queryplan bepaalt bijvoorbeeld hoe een eventuele join wordt afgewikkeld.

Paragraaf 13.3
Indexen (interne gegevensstructuren voor snelle toegang tot records) vormen een van de middelen voor de optimizer om de performance te verbeteren. Bij snelle toegang moet worden gedacht aan zoekprocessen of benadering van records in een bepaalde sorteervolgorde.

Op primaire en verwijssleutels wordt automatisch een *standaardindex* aangemaakt. Door de programmeur (of database administrator) kunnen indexen op andere kolommen (of kolomcombinaties) worden aangemaakt. In principe bepaalt de optimizer of een index wordt gebruikt als onderdeel van het queryplan. Het queryplan is echter aanpasbaar. Wordt geen index gebruikt, dan wordt de tabel in kwestie rij voor rij doorlopen (een full table scan).

De *lineaire-lijstindex* en de (meestal snellere) *binaire-boomindex* (bomen met twee takken per knooppunt) zijn voorbeelden van gegevensstructuren waarmee een index intern kan worden gerealiseerd. In de praktijk worden vaak bomen gebruikt met meer dan twee takken per knooppunt, die niet al te diep zijn.

Omdat het bijhouden van een index tijd kost en optimizers soms verkeerde keuzes maken, is het nut van een index niet bij voorbaat zeker. Metingen moeten uitwijzen of er echt sprake is van winst.

Een *unieke index* heeft als neveneffect een uniciteitscontrole.

Een van de (statistische) gegevens op grond waarvan de optimizer bepaalt of een index wordt gebruikt, is de *selectiviteit* van de index. Deze is hoog wanneer de index relatief veel verschillende waarden bevat. Een index met lage selectiviteit is doorgaans weinig zinvol.

Voor goede prestaties moet een indexboom (min of meer) *gebalanceerd* zijn, wat wil zeggen dat de takken ongeveer even diep zijn. Bij scheefgegroeide bomen kan onderhoud noodzakelijk zijn, bijvoorbeeld door de index inactief te maken en opnieuw te activeren.

Paragraaf 13.4
Bij slechte performance van een query kan het nuttig zijn op zoek te gaan naar een equivalente query die het beter doet. Lijstjes van wat goed is en wat niet, zijn vaak dialect- of zelfs versiespecifiek.

Paragraaf 13.5
Aanpassing van het (technisch) databaseontwerp kan soms de performance van query's ten goede komen. Bijvoorbeeld de bewuste introductie van (bewaakte) redundantie. Tot de mogelijkheden behoren: het opslaan van statistische gegevens en denormaliseren van de tabelstructuur. Ook de keuze van sleutels (hun datatype en wel of geen kunstmatige sleutels introduceren) kan van grote invloed zijn.

Hoofdstuk 14

Paragraaf 14.2
Bij het oplossen van een `select`-probleem zijn belangrijke vragen:
- *Vraag 1*: Waarover wordt informatie gevraagd?
- *Vraag 2*: Uit welke tabellen is de gevraagde informatie afkomstig?
- *Vraag 3*: In welke tabellen staan de gegevens die een rol spelen?

Uit vraag 1 is af te leiden welke tabel het startpunt is van het navigatiepad. Vraag 2 als uitgangspunt van een oplossingsstrategie leidt veelal tot een oplossing waarbij wordt genavigeerd via een subselect, vraag 3 leidt eerder tot joinnavigatie. In de oplossingen van complexe problemen kunnen beide navigatievormen worden gecombineerd. Het oplossen van problemen via vraag-en-antwoord kan belangrijk bijdragen aan het reduceren van complexe problemen tot kleinere deelproblemen en aan heldere code. Een belangrijk element is het formuleren van deelproblemen in natuurlijke taal, die in de eindoplossing zichtbaar blijven via commentaarcode.

Paragraaf 14.3
Bij het oplossen van complexe problemen wordt een stappenplan aanbevolen:
- *Stap 1*: ga na of de vraagstelling 100% duidelijk is.
- *Stap 2*: ga na hoe je het probleem handmatig zou oplossen.
- *Stap 3*: overweeg alternatieve aanpakken.
- *Stap 4*: transformeer het probleem stapsgewijs (vraag-en-antwoord) naar SQL.

Het stapsgewijs oplossen houdt in: elk probleem of deelprobleem volledig en correct oplossen, waarbij natuurlijke taal mag worden gebruikt voor de (exacte!) formulering van deelproblemen, zoals besproken in paragraaf 14.2.

Hoofdstuk 15

Paragraaf 15.1
Een *transactie* is een reeks databasebewerkingen die een logische eenheid vormen en die of helemaal wordt uitgevoerd (`commit`) of helemaal niet (`rollback`). In het *impliciete transactiemodel* wordt steeds impliciet een transactie gestart, waarbij de `commit` wordt uitgesteld tot de gebruiker die zelf geeft (of een `rollback`). In het *expliciete transactiemodel* wordt ieder los statement direct gecommit en wordt een samengestelde transactie pas gestart na een expliciet `start transaction`-statement.

Paragraaf 15.2
Transacties vragen om speciale acties van het rdbms in het geval van *concurrency* en *recovery*.
Concurrente transacties zijn transacties die tegelijkertijd met hetzelfde deel van een database bezig zijn. Het rdbms moet ervoor zorgen dat concurrente transacties geen problemen voor elkaar veroorzaken, zonder dat dit de performance aantast. Daartoe past het rdbms *concurrency control* toe: meestal *locking* (het 'op slot' zetten van een deel van een database tot de huidige transactie er klaar mee is), *multiversion concurrency control* (het bewaren van oudere versies van data, die gebruikt kunnen worden door de ene transactie terwijl een ander nog bezig is een nieuwe versie van die data te maken) of *optimistic concurrency control* (het rdbms onderneemt pas actie als concurrente transacties daadwerkelijk een probleem krijgen).
Een database waarin transacties zijn toegestaan, moet voldoen aan de ACID-eisen: *atomicity*, *consistency*, *isolation* en *durability*.

Paragraaf 15.3
Als concurrente transacties zijn toegestaan kunnen er, als er geen adequate concurrency control is, vier soorten problemen optreden: *lost updates*, *dirty reads*, *non-repeatable reads* en *phantoms*.
Een transactie T heeft een *lost update* als het resultaat van een `update` binnen T tijdens de verwerking van T door een concurrente transactie ongedaan wordt gemaakt. Een transactie T doet een *dirty read* als T ongecommitte gegevens

gebruikt die door een concurrente transactie zijn neergezet. Een transactie T doet een *non-repeatable read* als T tweemaal dezelfde rij denkt te lezen, terwijl een concurrente transactie die rij intussen heeft gewijzigd en gecommit. Een transactie T ziet een *phantom row* verschijnen als T tweemaal dezelfde verzameling rijen denkt te lezen, terwijl een concurrente transactie intussen een rij aan die verzameling heeft toegevoegd en gecommit.

Paragraaf 15.4
Een manier om concurrency control te regelen is via *isolation levels*. Per transactie kan dan worden aangegeven wat die transactie mag zien van concurrente transacties: helemaal niets (serializable), alleen gecommitte inserts (repeatable read), alleen gecommitte updates, deletes en inserts (read committed) of alle gecommitte en ongecommitte wijzigingen (read uncommitted).
In alle gevallen veroorzaakt een poging tot wijzigen van een rij door een transactie een update-conflict (lock conflict) als een concurrente transactie nog bezig is met die rij (nog geen commit of rollback).
Bovengenoemde maatregelen zijn zo beschreven in de SQL-standaard; elk rdbms beslist zelf in hoeverre het zich hieraan houdt.

Hoofdstuk 16

Paragrafen 16.1 t/m 16.3
Stored procedures en triggers zijn programmaatjes, geschreven in een procedurele (3GL) *triggertaal*. Ze zijn opgeslagen in de database en actief aan de serverkant. Triggertalen zijn te beschouwen als dialectspecifieke, procedurele uitbreidingen van SQL. Door de typische procedurele elementen (keuze- en herhalingsstructuren) is het mogelijk er verfijnde controles en acties mee te implementeren.

Een *trigger* wordt aangeroepen bij het optreden van een bepaalde gebeurtenis (*event*). Een before-insert-trigger bijvoorbeeld wordt actief bij een poging tot invoegen van een rij in een specifieke tabel. Een after-insert-trigger wordt actief na het daadwerkelijk invoegen. Analoge triggers kunnen worden gecreëerd voor delete- en update-events.
Via de *contextvariabele* `new` zijn in de code van een update- of insert-trigger de nieuwe kolomwaarden beschikbaar. Analoog zijn via `old` de oude kolomwaarden beschikbaar in de code van een update- of delete-trigger.

Een *stored procedure* moet, via een `execute`-statement, expliciet worden aangeroepen, vanuit een trigger, een andere stored procedure of een applicatie. Vaak wordt in een stored procedure de eigenlijke actie geprogrammeerd, terwijl triggers alleen maar zorgen dat de stored procedure op het juiste moment wordt aangeroepen.

De verwerking van een trigger of stored procedure kan worden afgebroken door het 'opgooien' van een *exception* (een databaseobject met een naam en een geassocieerde foutmelding). Bij een trigger van het before-type wordt hiermee tevens de beoogde actie (insert, delete of update) afgebroken. Eventueel kan de verwerking worden voortgezet door het uitvoeren van alternatieve code (*vangen* van de exception) in een `when`-clausule. Via dit mechanisme (*exception handling*) kunnen regels worden geïmplementeerd die niet via een constraint kunnen worden afgedwongen.

Paragraaf 16.4
De *executable procedures* in dit hoofdstuk 'doen alleen maar wat' en retourneren geen waarde (zoals een functie). Men kan ook *select-procedures* creëren die wel een waarde teruggeven. Die waarde kan zelfs een tabel zijn. Dit opent vele mogelijkheden voor procedurele rijgewijze verwerking van tabellen.
Applicatieconstraints zijn regels die actief zijn aan de clientkant. Ze horen tot het domein van applicatieontwikkeling. *Load balancing* (de keuze welke regels aan de server- en welke aan de clientkant moeten worden geïmplementeerd) is echter ook een databaseprobleem.

Hoofdstuk 17

Paragraaf 17.1
Gegevens om gegevensstructuren vast te leggen, heten *metagegevens*. Deze zijn zelf gestructureerd volgens een *metastructuur*. De metastructuur van relationele schema's beschrijft wat deze schema's gemeenschappelijke hebben. Die schema's zijn op hun beurt de structuurbeschrijving van een concrete relationele database.
Tegenover het *metaniveau* van de structuurbeschrijving staat het *objectniveau* van concrete relationele schema's.

De metastructuur van relationele schema's is zelf ook relationeel. Metagegevens worden dus, net als gegevens van het objectniveau, vastgelegd in relaties (tabellen). Deze vormen samen de *data dictionary* of *systeemcatalogus*. Zo is er een metatabel met gegevens over alle tabellen van de database, inclusief de metatabellen zelf.

De structuur van de data dictionary is – afgezien van het feit dat deze altijd relationeel is – dialectspecifiek.
Paragrafen 17.2-17.3
Om te illustreren hoe een data dictionary kan zijn opgezet, is in deze paragrafen een deel van de data dictionary van Firebird behandeld, aan de hand van een eenvoudige gebruikersdatabase. Hieruit komt naar voren:
- Elke database heeft een eigen data dictionary.
- Elke data dictionary heeft dezelfde tabellen.
- De populatie van de data dictionary wordt deels bepaald door structuur van de gebruikersdatabase, voor een ander deel door de data dictionary zelf; dit laatste deel van de populatie is dus in elke data dictionary hetzelfde.
- Het effect van een DDL-statement is in principe equivalent met een verzameling DML-statements uitgevoerd op de data dictionary. Beperkings- of gedragsregels, werkzaam op een data dictionary (bijvoorbeeld via triggers), en autorisatiebeperkingen zullen de mogelijkheden in de praktijk beperken of zelfs tot nul reduceren.

Paragraaf 17.4
Metaviews of *systeemviews* zijn views op de data dictionary. Deze kunnen dienen om gebruikers op een veilige manier bepaalde rechten op de data dictionary te geven, bijvoorbeeld om objecten waarvan zij eigenaar zijn, in te zien. Een ander doel kan zijn: het bieden van een gestandaardiseerde, dus dialectonafhankelijke manier om gegevens te lezen uit de data dictionary. ANSI/ISO heeft een aantal van dergelijke metaviews gespecificeerd.

Paragraaf 17.5
Elke gebruikersdatabase (database van het objectniveau) is op te vatten als een uitbreiding van zijn data dictionary. Aangezien de data dictionary niet alleen de structuur van de gebruikersdatabase beschrijft, maar ook zijn eigen structuur, bestaat er geen apart *meta-metaniveau*: mochten we daar al van willen spreken, dan valt dit samen met het metaniveau. Dit correspondeert met het feit dat de data dictionary zelf ook een 'gewone' database is en als zodanig tot het objectniveau behoort: 'metaniveau' en 'objectniveau' zijn relatieve begrippen.