Vragen Databases

Hoofdstuk 1

1. Data: gegeven feiten
   Informatie: gegeven feiten + de betekenis, de context.
   Nuttig: huidige databanksystemen zijn enkel in staat de data op te slaan (de informatie is vager, bevat onzekerheden, fouten ...) -- Men probeert hier een mouw aan te passen met bijvoorbeeld vage databanken.
2. Database: collectie van persistente data
   Databasesysteem: computersysteem om databases te beheren
   Databasemanagementsysteem: softwareonderdeel van het databasesysteem
3. Opsplitsing in lagen zorgt ervoor dat de technische details van de fysieke zaken in het databasesysteem niet gekend moeten worden door gebruikers.
   Het toevoegen van een derde laag (de externe laag) zorgt er bovendien voor dat data 'op maat' gepresenteerd kan worden, afhankelijk van de gebruiker/het programma.
   Opsplitsing in lagen zorgt ook voor een makkelijker te onderhouden systeem doordat aanpassingen in een bepaalde laag niet gepropageerd moeten worden doorheen alle lagen (merk op dat de mapping tussen de aangepaste laag en die erboven wel aangepast moet worden, maar dat is het dan ook).
   Intern, logisch, extern.
4. Hardware, data en software. Er zijn ook nog de gebruikers.
5. Primair geheugen: snelste geheugen, geheugen waarop rechtstreeks de queries worden uitgevoerd.
   Secundair geheugen: bevat volledige databank.
   Tertiair geheugen: archivering, back-ups ...
   Databasebuffers: inladen buffer ipv 1 enkel record is efficiënter (queries worden toch normaal gezien op een boel records van een recordtype uitgevoerd).
   2 databasebuffers: parallel inlezen, en wegschrijven.
6. Record: collectie van velden die data kunnen bevatten.
   Er zijn records met vaste of variabele lengte.
   Die met vaste lengte zijn eenvoudig om op te slaan: Fileheader bekijken om te zien waar alle delen van de record zijn opgeslaan.
   Bij records met variabele lengte maakt men gebruik van separatorkarakters (einde_veld, einde_waarde en einde_record), zie figuur 1.8 op p.22.
7. Datawarehouse: bevat de geschiedenis van de database -- gebruikt voor data-analyse.
   Er kunnen verschillende dimensies aan de opslag van de data toegevoegd worden, een ervan is de tijd, dat is dan het bijhouden van de geschiedenis, maar het is er niet toe beperkt.
   Dit kan bijvoorbeeld ook de samenvoeging van verschillende databases betreffen ipv dezelfde op verschillende tijdstippen
   Datamining: zoeken van patronen, verbanden in de data.
8. Gebruikersprofielen: bepalen voor elke gebruiker welke bevoegdheden die heeft.
   Data-administrator: beslist welke data waar moet opgeslaan worden in welk formaat. Hij bepaalt ook welke gebruikersprofielen er zijn, wie welk gebruikersprofiel krijgt, welke beveiliging gebruikt wordt, hoe de data verwerkt wordt en wie er voorrang krijgt in het geval van toegangsconflicten.

Database-administrator: technische zaken, implementatie en onderhoud database, herstel bij falen, enz.

9. Databasedefinitie: structuur en semantische regels vastleggen, integriteit garanderen. Incorrectheid van data moet vermeden worden!
   Databasemanipulatie: toevoegen, verwijderen, aanpassen en doorzoeken. Deze moeten efficiënt gebeuren.
   Databaseconstructie: database efficiënt bijhouden in geheugen, eindgebruikers mogen niets merken van fysieke opslagdetails.
10. Anders moet je iedere keer wachten tot iemand anders klaar is.
    Ideaal, dan moet je die concurrency-problemen niet meer oplossen :-p
11. Beveiligen tegen ongeoorloofd gebruik (Trudy!) en fysiek falen van databasesysteem en ook tegen goedbedoeld gebruik dat verkeerd loopt (dus restricties opleggen aan de gegevens die ingegeven mogen worden).
    Je wil niet dat gegevens verloren gaan of in de verkeerde handen terechtkomen.
12. Fysieke dataopslag is heel systeemspecifiek, en wordt dus beter buiten het dbms gehouden.
    Nut interne laag: organiseren van data in records, en die records in bestanden.
    De interne laag levert dus de abstractielaag die nodig is om de systeemspecifieke dataopslag buiten het ontwerp van het systeem te houden.
13. Indices bevatten een waarde van het veld waarop de index gedefinieerd is, en een referentie naar het veld. Omdat indices een gesorteerde lijst van de velden aanleggen, kan je dan sneller werken.
14. Logische laag: handig voor abstracte voorstelling van alle entiteiten, verwantschappen, operatoren, integriteitsbeperkingen en dergelijke. De interne laag geeft nog altijd redelijk wat details over het fysieke niveau waar je eigenlijk niet mee geconfronteerd wil worden.

    Externe laag: verschillende zichtbare zaken voor verschillende types gebruikers (views). Meerdere voorstellingen van dezelfde data, het voorstellen van afgeleide data! (kan niet op het logische niveau)
15. Onderste laag aanpassen zonder dat de bovenste laag er last van heeft.
    Voorbeeld: overschakelen van harde schijf naar flash, de logische en externe laag zullen daar niks van merken.

    We hebben fysieke dataonafhankelijkheid door het toevoegen van de logische laag. Aanpassingen op het fysieke niveau worden dan opgevangen door de mapping tussen de interne en de logische laag

    We hebben logische dataonafhankelijkheid door het toevoegen van de externe laag. Dit wil zeggen dat we de logische beschrijving van de database kunnen aanpassen (en de logische-externe mappings natuurlijk) zonder dat de gebruikers (die met views werken) daar ook maar iets van merken.
16. -Hoeveel dbms-faciliteiten zijn nodig?
    -Wat is het budget? (kan wel opgelost worden met systemen als MySQL en PostgreSQL)
    -Hoe snel moet alles gaan?
    -Heeft het bedrijf een competent persoon in dienst die tijd kan spenderen om database-administrator te zijn (kan uitmaken bij klein bedrijf!)?
    -is er al iemand die met het databasesysteem kan werken of moet er nog iemand opgeleid worden?

-opslagcapaciteit van het toestel waarop de toepassing draait, vb gsm, gameboy (heeft ook wel een deel met het budget te maken)

17. Mijn ma haar kalender.
    save games voor een spel, steekt da gewoon in nen folder (zeker op nen gameboy bv., als ge al meerdere saves toelaat) <-- gameboyfreak!

Hoofdstuk 2

1. Datamodel is verzameling van voorschriften en regels die het mogelijk maken om de structuur en het gedrag te beschrijven van data die in bepaalde software voorkomen (dit kan zich beperken tot de specificatie van de datatypes die de software ondersteunt).

   Een databasemodel is een verzameling van voorschriften en regels die het mogelijk maken om zowel de structuur, de beperkingen voor integriteit en beveiliging, als het gedrag van een database te beschrijven op het niveau van de logische laag uit een 'drielagen'-architectuur voor databasemanagementsystemen.

   Databaseschema is het resultaat van het gebruik van een databasemodel in een specifieke situtatie, met andere woorden: het schema dat de data beschrijft die bij die situatie relevant is.
2. De operationele modellen omschrijven de structuur van de data, maar niet de bewerkingen (vb. boomstructuur, graafstructuur). Opzoeken en aanpassen van data gebeurt dmv specifieke operatoren in een hosttaal. Een voordeel van dit model is dat dit zeer efficiënt kan gebeuren.
   We hebben er twee besproken, het hiërarchisch (boomstructuur) en het netwerkmodel (graafstructuur).

   Relationele modellen voegen ook een wiskundige onderbouwing toe voor de operatoren. Bovendien geldt bij relationele modellen dat de tabellen uit atomaire waarden zijn opgebouwd. Het opzoeken/aanpassen van data gebeurt niet langer vanuit een hosttaal, maar mbv een taal eigen aan het model, SQL.

   Bij semantische modellen voegt men ook complexe objecten toe aan de database, met dingen als encapsulation en overerving.
   Voorbeelden: hiërarchisch, relationeel, objectgeoriënteerd
3. In een boomstructuur (eventueel met virtuele ouderknopen). De informatie wordt fysiek opgeslagen volgens het nabijheidsprincipe (cfr. geïntegreerde data).
4. Virtuele ouderknopen worden ingevoegd in de boomstructuur, maar verwijzen niet naar de werkelijke ouder in de boom (in het hiërarchisch databasemodel mag een knoop slechts 1 ouder hebben). Deze worden gebruikt om te verwijzen naar een andere tabel die gerelateerd is met de huidige tabel. (vb. schilderij heeft ouderknoop kunstenaar, en virtuele ouderknoop eigenaar)
5. Hiërarchische recordsequenties zijn een ordening van de data in een hiërarchisch databasemodel door de records in de boom in pre-order te doorlopen. Ze zijn nodig om de 'nabijheid' van knopen in de boom duidelijk te maken.
6. Commando's in de hosttaal (programmeertaal zoals Pascal). Er zijn softwarebibliotheken van de databaseontwikkelaar nodig.
   Nadeel: de verwerking van een databasemanipulatie gebeurt record per record. Programmeur moet eerst hostvariabelen declareren vooraleer hij een operator oproept.

Je kan enkel de programmeertalen gebruiken waarvoor een bibliotheek ontwikkeld is.
Voordeel: je kan gelijk welke programmeertaal gebruiken waarvoor een bibliotheek ontwikkeld is. Het oproepen van variabelen is direct en efficiënt.

7. In het netwerkmodel is men niet langer tot een boomstructuur om de data weer te geven, maar kan men gebruik maken van een algemene graafstructuur.

   Voordeel: ingewikkeldere relaties (want algemene graaf i.p.v. boom) mogelijk en meer verbindingen, waardoor datatoegang vaak veel efficiënter is dan bij het hiërarchisch model.
   Nadeel: geen 'logische nabijheid' meer. Ingewikkelder.
8. Waarden in een tabel die niet meer opgesplitst kunnen worden.
   Beperkingen: moeilijk om complexe data voor te stellen.
9. Kandidaatsleutels en vreemde sleutels.
10. -Objectgeoriënteerd steunt puur op de notie van objecten, meteen volledig ingebouwd, terwijl het er bij objectrelationeel als het ware 'opgeplakt' is.
    -Objectrelationeel heeft als voordeel dat de relationele structuur zorgt voor een lagere instapdrempel, betrekkelijke eenvoud en wiskundige hanteerbaarheid.
    -Objectrelationeel heeft een standaard, terwijl er voor puur objectgeorienteerde dbs nog altijd geen algemeen aanvaarde standaard is.
    -Objectrelationeel geleverd door bedrijven die het systeem zullen onderhouden en updaten, objectgeorienteerd minder gevestigde waarde, veel minder zekerheid.
11. Er treedt nog steeds een informatieverlies op bij omzetten van gegevens naar datastructuren van een programmeertaal of omgekeerd.
    Bij objectgeoriënteerde databasemodellen is dit verlies nog niet weggewerkt, bijvoorbeeld 'vage data' kan nog niet voorgesteld worden.
    Ook zekere informatie kan dikwijls nog niet perfect gemodelleerd worden in een (objectgeorienteerde) db omdat er veel verschillende interpretaties zijn voor het concept 'object'.
12. Voordeel: veel minder kans op vendor lock-in + goedkoop.
    Nadeel: kan minder goed zijn op vlak van technische ondersteuning en aanleercurve.

Hoofdstuk 3

1. Informatievergaring, conceptueel ontwerp, logisch ontwerp, fysiek ontwerp
   Informatievergaring: vragen aan de toekomstige gebruikers, documenten doorzoeken, ...
   Conceptueel ontwerp: EER-diagram + functionele beschrijving
   Logisch ontwerp: omzetten naar databaseschema + gedragsspecificaties
   Fysisch ontwerp: DDL-scripts + implementatie gedrag
2. Programma's waarmee men EER-diagrammen kan tekenen (en soms automatisch omzetten naar DDL-scripts).
   Voordelen: interactief, makkelijker om aan te passen, ziet er grafisch beter uit dan een schets
   Nadelen: ondersteunen soms niet alle ontwerpfasen, genereren geen geoptimaliseerde DDL-scripts, zijn niet compatibel met andere CASE-tools, kunnen soms geen n-aire relaties weergeven.
3. Entiteit is een voorwerp dat in de reële wereld bestaat.
   Een entiteittype karakteriseert een collectie van entiteiten en wordt gekenmerkt door een naam en een verzameling van attributen
4. Attribuut: gemeenschappelijke karakteristieken van 1 entiteitstype
   Relatietype: verwantschap tussen meerdere entiteitstypes
   Een attribuut is meer een 'eigenschap' van een entiteit, terwijl een relatietype gaat over het verband tussen twee entiteiten(types)

5. Enkelwaardige attributen: attribuut dat slechts 1 waarde mag aannemen op ieder tijdstip. Vb. plaats waar een persoon zich op dit moment bevindt.
   Meerwaardige attributen: attribuut dat meerdere waarden kan aannemen op hetzelfde tijdstip. Het meerwaardig attribuut vertegenwoordigt dus een set van atomaire waarden. Vb. lijst van kleuren die gebruikt zijn in een bepaald schilderij.
6. Een samengesteld attribuut bestaat uit meerdere andere attributen.
   Samengesteld: adres bestaat uit straat, huisnummer, stad, land.
   Enkelvoudig: naam van een boek.
7. Omdat afgeleide data niet mag opgeslaan worden in een database: leeftijd mag men niet opslaan in een database, omdat deze ieder jaar verandert. Het is beter om leeftijd met een query af te leiden uit een geboortedatum, anders is er meer kans op inconsistenties.
8. Om aan te duiden dat een bepaald entiteitstype volledig afhankelijk is van een ander entiteitstype, en niet voorkomt zonder het een definiërende entiteit van het andere entiteitstype.
   Deze zijn nodig omdat het een bestaande situatie is en die dus gemodelleerd moet kunnen worden. Een schilderij kan bijvoorbeeld niet bestaan zonder een artiest.
9. Kardinaliteit: vastleggen hoeveel entiteiten van een type mogen voorkomen in een relatie.
   Participatie: vastleggen of het nodig is dat minstens 1 entiteit van een type MOET voorkomen in een relatie.
10. Entiteittype dat een een deel van de entiteiten van een bepaald type karakteriseert. Nut: het is handiger dan de entiteitstypes als totaal onafhankelijk te beschouwen, omdat een deel van de attributen gemeenschappelijk zijn.
    Ook is het makkelijker om een subgroep te vinden.
11. Subtypes hebben alle attributen van het supertype ook, zonder dat we ze moeten kopiëren naar die subtypes.
    Dit is handig wanneer er een aantal verschillende entiteiten moeten voorgesteld worden die heel wat attributen gemeenschappelijk hebben.
12. Men heeft de mogelijkheid te bepalen tot welk subtype een gegeven entiteit van het supertype behoort met behulp van een extra attribuut dat het subtype aangeeft.

    ?Hoe werkt men als de subtypes niet geconditioneerd zijn: ???
    We kunnen weergeven of er totale of partiële participatie is van het supertype en ook of de subtypes disjunct of overlappend zijn.

    opm: de vorming van Supertype/subtype-netwerken en Supertype/subtype-hierarchieen + beperking op meervoudige overerving.
13. Speciaal subtype met verschillende supertypes.
    Om entiteiten van de supertypes te groeperen.
    Bij een categorie moet een entiteit niet van alle supertypes subtype zijn.
    voorbeelden:
    categorie: categorie=eigenaar, supertypes={bedrijf, museum, persoon}
    gewoon sub-type: sub-type=juweel, sypertypes={outfit, kunstwerk}
14. Een entiteit van een categorie erft enkel over van het supertype waartoe de entiteit behoort (dus niet van alle supertypes). Nut: We gebruiken categorieën om de entiteiten van de supertypes te groeperen.

Extra's:
domeinanalyse: opzoeken van achtergrondinformatie om de juiste betekenis en context van de data te achterhalen.

connection trap: ternair relatietype != 3 x binair relatietype (opl. gebruik centraal een zwak

entiteittype)

Hoofdstuk 4

1. Een datatype dat, relationeel gezien, niet verder opsplitsbaar is. Niet atomair: datum, adres, naam.
   Atomaire gegevens kunnen niet samengesteld of meerwaardig zijn, dus is het moeilijker complexe gegevens op te slaan.
   Voordeel: eenvoudiger.
2. Het domein van een datatype specifieert enkel de toegelaten atomaire waarden, terwijl het datatype ook de operatoren specifieert.
3. Uit een relatieschema en een extentie.
   -Geen dubbele tuples mogelijk
   -Niet geordende tuples
   -Niet geordende attributen
   -Atomaire attribuutwaarden
   Schema interpreteren als de declaratie van een welbepaalde veronderstelling (worst omschrijving ever!) en tuple uit extentie interpreteren als 1 bepaalde instantie van deze veronderstelling.
4. Catalogus bevat metadata.
   Catalogus kan enkel geraadpleegd worden door gebruikers, niet aangepast.

   ?Hoe kan ermee gewerkt worden???
   De catalogus wordt meestal bevraagd om het databaseschema te achterhalen om dan querries op te kunnen stellen.
   (impliceert natuurlijk dat het databaseschema niet zomaar voorhanden is, en eigenlijk, als het voorhanden is, dan is het hoogstwss niet volledig betrouwbaar, zeker als je een database van iemand anders krijgt, en nog zekerder als die al een tijdje bestaat)
   Daarnaast kan je ook views definiëren.
   Er wordt ook informatie ivm gebruikers en prestatie bijgehouden.
5. Views zijn benoemde, virtuele relaties, afgeleid van gebruiker/systeem-gedefinieerde relaties of andere views.
   Nut:
   -logische data-onafhankelijkheid
   -nieuwe data afleiden
   -kortere queries
   -voorzien in individuele gebruikersbehoeften

   Zoeken: analoog aan basisrelaties.
   Aanpassen: view moet aanpasbaar zijn, manipulatie mag niet conflicteren met definitie van view, geen manipulaties uitvoeren op afgeleide data.
6. Een index is een tabel waarin de attributen waarop de index gedefiniëerd is geordend worden, volgorde bepaald door indexdefinitie, en waarbij dan in een extra kolom een (logische of fysieke) pointer naar de data bewaard wordt.

   exacte def:
   Een index over n attributen van een basisrelatie R kan worden omschreven als een geordende lijst van (n+1)-tuples. Voor elk tuple t uit de extentie van R is er een (n+1)-tuple opgenomen in de lijst. Dit (n+1)-tuple is opgebouwd uit de n beschouwde attribuutwaarden van t en een referentie naar t. De (n+1)-tuples zijn geordend op basis van de n attribuutwaarden, volgens

een opgegeven volgorde.

Voordelen: verbetering van prestaties: zoeken wordt verbeterd door vb. binaire zoekalgoritmes
Nadelen: Aanpassingen, toevoegingen en verwijderingen duren langer.

7. Onbekende informatie en niet-gedefinieerde informatie (vb: 'prijs onbekend' en 'schilderij niet te koop').

   Waarde die correspondeert met 'ontbrekend'.

   Impact: er wordt een waarde toegevoegd aan het domein van de datatypes, namelijk null

   Problemen: logica kan probleem ondervinden (is nu drie-waardig, en 'wet van uitgesloten derde' is niet meer geldig), vb. null of niet null = null
   vb: (alles voor 1800 gemaakt) + (alles niet voor 1800 gemaakt) != alles (doordat null-waarden hetzelfde als 'vals' behandeld worden bij querries)
8. Default-waarde: als geen waarde wordt opgegeven, gebruikt men een standaard benaderingswaarde.
   Nadeel: slechts een benadering, dus verkeerde resultaten mogelijk.
   Voordeel: er is altijd een waarde om mee te werken, kan goede benadering zijn, kan info weergeven (vb. 'geen waarde'-string)
9. Kandidaatsleutel: uniek en irreducibel beest
   exact:
   Als K een deelverzameling is van de verzameling van alle attributen van een gegeven basisrelatie R, dan is K een kandidaatsleutel voor R als en slechts als voldaan is aan:
   1. De uniciteiteigenschap: geen enkele legale extentie van R bevat twee tuples met dezelfde waarden voor alle attributen uit K.
   2. De irreducibiliteiteigenschap: wanneer uit K attributen worden weggelaten, mag nie meer voldaanzijn aan de uniciteigenschap.

   Vreemde sleutel: beest van de buren.
   exact:
   Een vreemde sleutel F van een basisrelatie R2 is een verzameling van attributen van R2, waarvoor geldt dat:
   1. Er bestaat een basisrelatie R1 (R1 niet noodzakelijk verschillend van R2) met een kandidaatsleutel K, die evenveel attributen bevat als F en waarbij er een één-op-ééncorrespondentie is tussen de attributen van K en de attributen van F, zodat corresponderende attributen dezelfde geassocieerde datatypes hebben.
   2. Op elk tijdstip komt elke reguliere waarde van F in R2 eveneens voor als waarde van in K in een tuple van R1.

   Om te verwijzen naar andere relaties. Zo kunnen verbanden tss verschillende relaties (tabellen) gelegd worden.
10. Cyclische referenties: vreemde sleutel in R1 verwijst naar R2, vreemde sleutel in R2 verwijst naar R1.
    Voorbeeld: een Eigenaar rijdt met een Voertuig, dat wordt hersteld door een Werknemer, die werkt voor een Eigenaar.
11. Elk van de reguliere waarden van een vreemde sleutel komt op elk moment voor als waarde van de corresponderende kandidaatsleutel.
12. Voorwaarde waaraan data op elk moment moet voldoen (om voor consistentie te zorgen).

Toestandsrestrictie vs. Transitierestrictie
Relatierestrictie vs. Databaserestrictie

13. Stored procedure is geavanceerder, voorgecompileerde groep bewerkingen.
14. Als een bepaald trigger event zich voordoet en de trigger condition evalueert als waar, wordt de stored procedure uitgevoerd.
    Een trigger bestaat uit: zijn naam, trigger event, trigger condition en triggered action.
15. Dat is een 'discussie'-vraag natuurlijk. Maar voor mij is het eigenlijk volwaardig relationeel, omdat:

    Het relationeel databasemodel wordt gekenmerkt door twee zaken, het is onderbouwt door de relationele algebra en het is een structureel model.
    De relationele algebra zegt niets over stored procedures en triggers.
    En het is perfect een structureel model (benadrukken van het concept 'abstracte structuur', onafhankelijk van de fysieke organisatie en opslag van de data), ook zonder stored procedures en triggers.

    Dus er is geen enkel probleem. Nu, wel problemen wss, maar niet om het volwaardig relationeel te noemen.
16. vereniging (unie), doorsnede (intersectie), verschil en cartesiaans product
    selectie (R where e), projectie (bepaalde kolommen 'selecteren' eigenlijk), join (inner en left outer), deling (alsof ge factoren uit een deling wegdeelt)

    vereninging, doorsnede en verschil moeten relaties van hetzelfde type zijn!

    ze zijn allemaal nodig veronderstel ik, iemand die het tegendeel beweert?
    R1 unie R2 = omega verschil ((omega verschil R1) unie (omega verschil R2))
17. De intersectie operator levert (op R1 en R2) de tuples die in zowel R1 als R2 voorkomen.
    Voorwaarde: R1 en R2 moeten een schema van hetzelfde type hebben.
18. Bij cartesiaans product worden alle samenvoegingen van tuples beschouwd, bij de join enkel tuples die gerelateerd zijn via dezelfde waarden voor de gemeenschappelijke attributen.
19. **blah**
20. De delingsoperator werkt op dezelfde manier alsof je factoren zou wegdelen uit een breuk/veelterm. Alleen tuples waaruit je iets weg hebt kunnen delen worden weergegeven, algemeen XY DIVIDEBY X = Y

    Deze operator is ingevoerd met het oog op de verwerking van vraagstellingen waarbij gebruik wordt gemaakt van de 'universele quantor' (of 'voor alle')
21. Elke operator werkt in op relaties en produceert een nieuwe relatie. Belangrijk: voor het samenstellen van operatoren.

Extra's:
typecompatibiliteit: 2 datatypes zijn typecompatibel als we de domeinwaarden van het ene datatype kunnen omzetten naar dominwaarden van het andere datatype en omgekeerd.

graad van een relatie = aantal attributen, de kardinaliteit is het aantal tuples

virtuele relatie: niet gematerialiseerde relatie (cfr. views)

querymodificatie: bij querrymodificatie wordt voor het uitvoeren van een querry op een view de definiërende expressie van de view in de originele querry gesubstitueerd en dan wordt de resulterende querry rechtstreeks op de basisrelaties uitgevoerd.

uniciteitindex: het systeem zal automatisch op bepaalde attributen (zoals keys) een uniciteitindex definiëren, precies om de uniciteit ervan af te dwingen aangezien dan geen twee keer dezelfde (combinatie van) attributen voor mogen komen

bij de relationele algebra hebben we ook de uitbreiding (eigenlijk om afgeleide attributen te kunnen appenden, maken), groepering en aggregatieoperatoren

Hoofdstuk 5

1. EER-diagram wordt omgezet naar databaseschema met omzettingsalgoritme.
   De functionele beschrijving wordt omgezet naar gedragspecificaties (mbv alternatieve sleutels, integriteitrestricties, stored procedures en triggers).
   In omgekeerde richting kan ook omzetting gebeuren (reverse engineering: EER-diagram actualiseren na aanpassen van het databaseschema).
2. CASE-tools (sommige) laten toe met de druk op een knop een databaseschema op te stellen met behulp van het EER-diagram.
3. A:verschillende basisrelaties, voor het supertype en de subtypes.
   Bij deze optie blijven alle bestaande basisrelaties behouden. Voeg de attributen van de primaire sleutel van het supertype als primaire sleutel toe aan de basisrelaties van alle subtypes. Deze attributen vormen in elk van deze basisrelaties tevens een vreemde sleutel.

   Voordeel: werkt voor alle mogelijkheden.
   Nadeel: veel basisrelaties.

   B:verschillende basisrelaties, enkel voor de subtypes.
   Bij deze optie worden alle attributen van het supertype toegevoegd aan de basisrelaties van alle subtypes. De primaire sleutel van de basisrelatie van het supertype wordt de primaire sleutel in elk van de basisrelaties van de subtypes. Na de toevoeging wordt de basisrelatie van het supertype verwijderd.

   Voordeel: Een relatie minder.
   Nadeel: enkel voor totale specialisatie/generalisatie met disjuncte subtypes.

   C:één basisrelatie met één typeattribuut.
   Bij deze optie worden alle attributen van de basisrelaties van alle subtypes toegevoegd aan de basisrelatie van het supertype. Daarenboven wordt er nog één extra typeattribuut toegevoegd, waarvan de waarde aangeeft of en tot welk subtype een bepaald tuple behoort. Na de toevoeging worden de basisrelaties van de subtypes verwijderd. De primaire sleutel van de basisrelatie van het supertype blijft ongewijzigd.

   Voordeel: slechts 1 relatie
   Nadeel: veel ontbrekende gegevens als er veel attributen zijn bij de subtypes, en kan enkel toegepast worden voor generalisatie/specialisatie met disjuncte subtypes.

   D:één basisrelatie met meerdere typeattributen.
   Bij deze optie worden opnieuw alle attributen van de basisrelaties van alle subtypes

toegevoegd aan de basisrelatie van het supertype. Nu worden er m typeattributen toegevoegd waarbij elk typeattribuut correspondeert met een subtype en aangeeft of een bepaald tuple al dan niet tot dat subtype behoort. Na de toevoeging worden de basisrelaties van de subtypes verwijderd. De primaire sleutel van de basisrelatie van het supertype blijft ongewijzigd.

Voordeel/Nadeel: idem als bij C, maar kan in meer gevallen toegepast worden.

4. Ja, voer optie 3B uit en voeg het typeattribuut toe aan het subtype.
   Bij gewone overerving van meerdere supertypes krijgt de het entiteittype sowieso alle attributen van alle supertypes, daar doen zich al niet veel moeilijkheden mee voor.
5. 1 op 1: Getrouwd met: het is niet logisch om de attributen van de man/vrouw toe te voegen aan de vrouw/man (logischer om als aparte entiteiten te houden).
   En het is ook niet logisch om de attributen van 'getrouwd' (vb. trouwdatum, gemeente ...) toe te voegen aan slechts 1 persoon.
   1 op N: analoog, maar we beschouwen ook polygame relaties.
6. Stel A is meervoudig, meerwaardig attribuut van relatie R, en is samengesteld uit B en C (enkelvoudig, enkelwaardig) en D (enkelvoudig, meerwaardig).
   Dan maken we een nieuwe basisrelatie A, met attributen B en C, plus een Foreign Key: de primaire sleutel van R. B en C vormen de primaire sleutel van A (als B op zich als uniciteit garandeert, laten we C vallen).
   We maken dan nog een nieuwe basisrelatie B, met als attribuut de waarde van B, en een Foreign Key die wijst naar de primaire sleutel van A. Primaire sleutel is het attribuut met de waarde van B.
7. Een verzameling van attributen Y is functioneel afhankelijk van een verzameling van attributen X als de waarden van de attributen van Y op elk moment uniek worden vastgelegd door de waarden van de attributen van X.
   Als de attribuutwaarden van X gekend zijn, zijn daardoor ook de attribuutwaarden van Y gekend. X wordt de determinant van de functionele afhankelijkheid genoemd.

   ?Waarom kunnen deze niet automatisch worden afgeleid uit de specifictie van een basisrelatie?
   De functionele afhankelijkheid kan zich binnen een enkele basisrelatie bevinden. Als er geen verwijzingen zijn dmv (foreign) keys dan is er volgens het model ook geen verband tss de informatie.
8. Irreducibel: functionele afhankelijkheid + X en Y zijn disjunct + er bestaat geen deelverzameling X' van X waarvoor geldt dat Y functioneel afhankelijk is van X'.
9. Een verzameling van attributen Y is meerwaardig functioneel afhankelijk van een verzameling van attributen X (genoteerd X->->Y) als de waarden van de attributen van X op elk moment een collectie met meerdere waarden voor de attributen van Y vastleggen.

   Een relatie staat in vierde normaalvorm indien ze in ‘Boyce-Codd’-normaalvorm staat en geen enkele meerwaardige functionele afhankelijkheid bevat, tenzij dit de enige afhankelijkheid is die voorkomt in de relatie.

Hoofdstuk 6

1. Databaseschema en een zo groot mogelijk deel van de gedragsspecificaties worden geïmplementeerd in het gekozen relationeel databasesysteem. Dit gebeurt mbv de DDL.
   Indien nodig moeten nog bepaalde gedragsspecificaties geimplementeerd worden in de externe software, dit hoort strict gezien niet bij het fysieke ontwerp.
   De nodige relaties worden aangemaakt, weer mbv DDL.
   Opvullen mbv DML.

forward engineering genoemd.
2. Uit de datadefinitietaal en de datamanipulatietaal, en nog extra instructies voor views, beveiligingsmechanismen en concurrency.
3. Implementaties van SQL door dbms-ontwerpers, maar wat 'aangepast'. De volledige standaard is niet noodzakelijk geimplementeerd, of extra mogelijkheden zijn toegevoegd om het gebruikersgemak te vergroten.
   Consequentie: opletten voor vendor lock-in.
4. Valideren en omzetten naar algebraïsche expressie, optimaliseren met behulp van queryplan, queryplan uitvoeren, resultaat teruggeven.

   DML-instructie -> SQL-Validatie -> algebraïsche expressie -> SQL-optimalisatie -> queryplan -> uitvoering van het querryplan door de querry-processor (regelt fysieke interactie met de data) -> resultaat teruggeven

5. Alle queries worden omgezet naar relationele expressies om aan querry-optimalisatie te doen.

   Toevoegen: insert into R1 values (v1,v2,...,vn) of R2 -> R1 union R2 (ev. met R2=(v1,v2,...,vn))

   Verwijderen: delete from R where c -> R except (R where c)

   Aanpassen: combinatie van nieuw toevoegen en oud verwijderen

   Opzoeken: zie figuur 6.7, p. 209
6. Equivalente expressie die efficiënter is wordt gezocht, en dan omgezet naar queryplan. Er wordt niet volledig gezocht, omdat het zoeken naar het queryplan zo lang kan duren dat er meer resources verspilt worden aan het zoeken, dan er gewonnen worden met de betere query.
7. zie p. 211, dees zijn ze:

   Het is belangrijk om de query goed te begrijpen en te weten wat de query verondersteld is te doen. Desnoods moet de query voor de analyse worden opgesplitst.

   Het is belangrijk om de relaties te kennen waarop de query inwerkt. Welke attributen zijn er? Welke datatypes worden gebruikt? Hoeveel tuples zijn er?

   Stel de query in vraag: Zijn alle gevraagde attributen nodig? Zijn er geen onnodige voorwaarden in de 'WHERE'- en 'HAVING'-gedeelten? Zijn er geen onnodige relaties opgenomen in het 'FROM'-gedeelte? Moeten de resultaten wel worden gesorteerd?

   Kan het 'WHERE'-gedeelte optimaal gebruikmaken van de beschikbare indexen? Bij sommige condities zoals 'IS NULL', 'OR', '<>', 'NOT', 'NOT IN', 'NOT EXISTS', 'NOT LIKE' en 'LIKE '%vrouw'' kunnen indexen niet optimaal worden gebruikt. Herschrijven van de query kan performantiewinst opleveren.

   Tracht te achterhalen hoe belangrijk de query is. Hoe dikwijls wordt de query uitgevoerd?

   Als de query gewoonlijk heel veel resultaten teruggeeft, kan het nuttig zijn om te overwegen

om de toepassing te herontwerpen.

Analyseer het queryplan. Als blijkt dat bepaalde indexen niet worden gebruikt in het queryplan, kan het nuttig zijn om het gebruik ervan af te dwingen. Sommige systemen laten het (de DBA) toe om expliciet bij een query op te geven dat een index moet worden gebruikt. Een andere mogelijkheid is om de query-instructie te herschrijven.

8. Invullen van een grafische template, die wordt omgezet naar een query. Voordeel: makkelijker. Nadee: niet alle SQL-queries kunnen zomaar omgezet worden naar QBE.

Hoofdstuk 7

1. Objecten zijn autonome entiteiten, hiervan zijn de structuur en het gedrag vastgelegd in het objecttype.
   Objecttypes worden opgebouwd uit literaaltype (structuurspecificatie),interfaces (gedragspecificatie) en klassen (structuur- en gedraspecificatie).
2. Inkapseling zorgt ervoor dat enkel de signatuur van operatoren zichtbaar is voor gebruikers, maar de methode blijft verborgen. Reden: gebruikers afschermen van implementatiedetails.
3. Overerving: deel van toestand en gedrag van andere objecttypes overnemen. Nut: specifiekere objecttypes opbouwen, en de specificaties van het algemenere objecttype hierbij hergebruiken.
4. Dier.maakGeluid()
   Koe.maakGeluid()
   Hierbij levert de operator een ander resultaat. --> operatoroverloading
   Stel dat we een deel Dieren (waaronder Koeien) in een verzameling steken, en één voor één op die dieren maakGeluid() uitvoeren, dan zal het systeem zelf afleiden dat Koe.maakGeluid() “Beeeuu!” oplevert. --> polymorfisme, late binding
5. Naamgevingsmechanisme: object kan persistent gemaakt worden door het een persistente naam te geven, uniek in de hele database.
   Bereikbaarheidsmechanisme: elk object waarnaartoe verwezen wordt vanuit een persistent object, wordt ook persistent.
6. Handig om objecten uniek te identificeren, en bovendien verandert de objectidentificator niet.
7. Literaaltypes modelleren structurele kenmerken. Objecttypes ook gedragskenmerken. Gevolg: alles moet persistent opgeslagen kunnen worden. Gevolg zou zijn dat ook aan literalen objectidentificatoren toegekend moeten worden.
8. Interfaces zijn bouwstenen om gedrag te modelleren.
   Klassen implementeren deze interfaces.
   Verschil klassen en objecttypes: objecttypes kunnen opgebouwd worden uit literalen, interfaces en klassen, terwijl klassen uit attributen, relaties en operatoren opgebouwd worden.
   Objecttypes zijn dus algemener.
9. Unidirectioneel: gerelateerde klasse opnemen als type van een attribuut.
   Bidirectioneel: 2 relaties, 1 in elke klasse.
   Nadeel unidirectioneel: geen referentiële integriteitscontrole
   voordeel unidirectioneel: eenvoudig
   Nadeel bidirectioneel: meer plaats innemen? meer werk
   voordeel bidirectioneel: referentiële integriteitscontrole
10. 'isa'-overerving (interfaces)

'extends'-overerving (klassen)

11. Entiteiten bij het relationeel model worden sowieso in de extentie geplaatst, objecten bij het objectgeoriënteerd modeel enkel als ze persistent zijn.
    En bij het relationeel model is een extentie natuurlijk mooi uit rijen van een tabel opgebouwd, terwijl dat bij ODMG wel niet mooi lineair opgeboud zal zijn.
12. Bij het ODMG-model is er slechts 1 sleutel, deze komt overeen met een kandidaatsleutel bij EER en relationeel.
    Ieder (persistent) object heeft sowieso een unieke objectidentificator.
13. Fysiek dataontwerp, toevoegen, aanpassen en verwijderen van objecten kunnen enkel via taalbindingen gebeuren.
    Voor objectbevraging is er ook nog altijd mogelijkheid tot directe DB toegang via de OQL.
14. OQL kan enkel gebruikt worden voor opzoeken van data (SQL ook voor manipulatie). OQL biedt wel meer flexibiliteit op vlak van zoekmogelijkheden en weergave van resultaten.
15. Nee. Nadeel: kans op inconsistentie.
    Voordeel: meer flexibiliteit/grotere semantische waarde/meer gebruikersgemak.
16. Met een variant van de 'CREATE TABLE'-instructie kan men een relatie aanmaken met de componenten van het tupletype als attributen en met een zelf gespecifieerde primaire sleutel.
17. Gebruik: om rechtstreeks van 1 tuple naar een ander tuple te navigeren.
    -identificator afleiden van primaire sleutel
    -identificator door systeem laten opbouwen
    kunt gebruik maken van -> om attributen ed aan te spreken van het tuple waarnaar verwezen wordt.
    bv. als je in klasse A werkt waarin een referentie naar schilderij staat: A.schilderij->waarde
18. Overerving bij definitie van een tupletype en overerving bij de aanmaak van een relatie.
19. Om multimediaobjecten in de database te integreren.
    Voordeel: sommige operatoren kunnen toegepast worden (vb. patroonvergelijking)
20. Het omzetten van objecttypes naar een relationeel databaseschema.

Extra's:
enumeratietype: enum kleur{rood, groen, blauw}

gestructureerd type: struct

iteratorvariabele: s in schilderij

referentieattribuut: bevat verwijzing naar ander object, tupletype, zie vraag 17

XML: Extensible Markup Language, beschrijvingstaal om zowel de structuur, als de inhoud van multimedia op een uniforme wijze te beschrijven.
SQL/XML uitbreiding voor SQL3, vb. uitbreiding SELECT

Hoofdstuk 8

1. Een API voor databasetoegang zorgt voor de verbinding of interface tussen de applicatie en het dbms. Het wordt door de applicatie gebruikt om instructies door te geven aan het dbms en wordt omgekeerd door het dbms gebruikt om resultaten en status- en foutcodes door te geven aan de applicatie.
2. Bij een ingebouwde API worden de instructies voor databasetoegang integraal ingebouwd in de applicatiecode. De uitvoering van de ‘verrijkte’ applicatiecode bewerkstelligt de interactie met het dbms, dat op zijn beurt verantwoordelijk is voor de correcte afhandeling van de instructies en de feitelijke databasetoegang.

3. De hosttaal is de taal waarvoor een API in de vorm van een softwarebibliotheek beschikbaar is gemaakt, zodat het mogelijk is in deze taal interactie te hebben met een dbms.
   Uiteindelijk, de taal waarin je aan het werken bent en van waaruit je ook toegang nodig hebt tot een database en daardoor die instructies 'inbed'.
4. Embedded SQL: SQL-instructies die ingebouwd zijn in de programmacode worden geprecompileerd naar een formaat dat het dbms kan uitvoeren.
   Voordeel: omzetting moet maar 1 keer gebeuren.
   Nadeel: databasespecifiek.

   De instructies die uitgevoerd worden liggen vast, tenzij je gebruik maakt van 'dynamische sql' zodat er toch late binding gebeurt, maar dan moet de instructie weer op het moment zelf omgezet worden natuurlijk.
5. Een cursor is een soort iteratorconstructie die bij embedded SQL toelaat om tuples één voor één op te halen en te verwerken.
   Stappen: DECLARE, OPEN, FETCH, CLOSE.
6. Dynamische SQL wordt niet geprecompileerd, waardoor meer flexibiliteit beschikbaar is (at runtime gedoe). Late binding is nodig juist om pas at runtime een omzetting uit te voeren.
7. Bij een call-level API gebeurt de databasetoegang via aparte software die communiceert met de applicatie en het dbms (dus instructies omzetten).
   Voordeel: geen precompilatie meer nodig, late binding laat meer mogelijkheden toe.
   Nadeel: omzetting gebeurt opnieuw bij elke uitvoering, dus is er vertraging.
8. Applicatie-interface: instructies van de applicatie verwerken en doorgeven aan de interne driverbeheerder.
   Driverbeheerder: selecteren van de juiste driver die overeenstemt met een bepaalde aanvraag en database.
   Driver: rechtstreekse communicatie naar database.
9. Environment: lijst toegekende verbindingen bij specifieke database
   Connection: specificieke verbinding
   Statement: specifieke instructie
10. Weinig verschil eigenlijk, beide maken gebruik van een fetch instructie die record per record ophaalt.
    Het is wel zo dat bij embedded SQL er gebruik gemaakt wordt van die 'cursor' en dat ge de cursor ophaalt en die dan in uw verschillende variabelen steekt, iets in de aard van: fetch schilderijcursor in schilderijstring,periode.
    Terwijl bij ODBC ge eerst zegt waarin de resultaten terecht gaan komen en ge dan gewoon uw handle fetcht.
11. Naast environment-, connection- en statement-records zijn bij SQL/CLI ook description-records toegevoegd, om extra informatie te krijgen over tuples.
12. OLE DB = object linking and embedding for databases
    ADO = ActiveX Data Objects

    Gebruikers communiceren met het OLE DB gedeelte via een call-level API. Dat gedeelte geeft informatie door aan COM-objecten (die bekend staan als 'data providers'), die dan communiceren met het dbms. Er kunnen ook extra COM-objecten toegevoegd worden om voor extra functionaliteit te zorgen. Deze COM-objecten heten 'service providers'.
    Verschil: COM-objecten hebben meer uiteenlopende functionaliteit dan drivers. Bij een relationele database zal een COM-object werken als een driver, bij een tekstbestand zal een beperkter aantal functies ondersteund worden.
    Er kan dus mbv verschillende drivers uit verschillende bronnen informatie gehaald worden.

figuur 8.8 p. 259 bekijken is geen slecht idee, veel overzichtelijker dan tekst

13. Connection: verbinding met databron, en verbindingsparameters initialiseren.
    Recordset: resultaten van zoekopdrachten ophalen.
    Command: zoekopdrachten met parameters of opdrachten die geen zoekopdracht zijn.
14. Pagina's die dynamisch worden opgebouwd als een gebruiker de gegevens uit een database/andere bron opvraagt.
    Databasetoegang gebeurt via de API of indirect via de webserver.

    Denk AsP (Active Server Pages), JSP (Java Server Pages) en PHP-scripttaal.

Extra's:
ADO.NET: deel van het .NET framwork van Microsoft, bevat OLE DB en ADO, er is ook een ODBC.NET

Hoofdstuk 9

1. diefstal van gegevens, schending van privacy, beschadiging van de database, sabotage van het databasesysteem
2. Uitgaan van een doordachte beveiligingsstrategie die niet alleen de kwetsbare punten van het databasesysteem identificeert en beschermt, maar ook vooruitkijkt en vastlegt wat moet worden ondernomen als de beveiliging wordt gecompromitteerd.
   Aandacht besteden aan alle componenten: gebruikers, hardware, software en data.
3. Toegangscontrole: zorgen dat enkel de juiste mensen in de database binnenmogen
   Toegangsbeperking: gebruikers mogen enkel bepaalde acties uitvoeren
   Auditbestanden: opmerken verdachte gebruikerspatronen, of achteraf de indringer vinden
   Versleuteltechnieken: zorgen dat de data, zelfs bij diefstal, niet gecompromitteerd wordt
   Views: enkel bepaalde data zichtbaar maken
4. Controle of de gebruiker is wie hij/zij beweert te zijn.
   Veiligste methode: oogscan + bloedscan + elektronische kaart +10 wachtwoorden. ;-)
   Nee, uiteindelijk is geen enkele methode 100% veilig (al wil ik niet weten op welke gruwelijke manier je bovenstaande lijst gaat omzeilen).
   Wachtwoorden hebben het nadeel dat de gebruikers ze moeten kunnen onthouden (en ze dus niet op een blaadje papier opschrijven), dat ze gekraakt kunnen worden.
   Zaken als een oogscan of bloedscan zijn omslachtig, en voor een elektronische kaart zou je een speciaal toestel nodig hebben.

   Uiteindelijk zou ik op zoek gaan naar een methode die het evenwicht zoekt: genoeg afschrikwekkend voor Trudy, maar toch niet te omslachtig voor de gebruikers (tenzij het werkelijk om iets van levensbelang gaat, dan mag het omslachtig zijn).
5. Als je vergeet een privilege toe te kennen, is dat vervelend voor de gebruiker, maar kan die altijd om extra privileges vragen.
   Als je vergeet een beperking toe te kennen, kan een gebruiker je databank beschadigen of er informatie uithalen.
6. Contextafhankelijk: privileges kunnen afhangen van een aantal voorwaarden, zoals de dag van de week.
   Verlaten: toegekend door persoon die zelf dat privilege niet meer heeft.
7. RESTRICT: enkel van huidige gebruiker privileges verwijderen
   CASCADE: ook van gebruikers die van huidige gebruiker privileges gekregen hebben
8. Voordeel: minder moeite (het is nogal omslachtig om voor iedere gebruiker apart de privileges te wijzigen)
   Nadeel: minder gedetailleerd (je verandert meteen privileges voor een hele groep)

9. Gebruikers kunnen enkel zaken lezen van eigen of lager beveiligingsniveau, en enkel schrijven naar eigen of hoger beveiligingsniveau.
10. Gebruikers krijgen enkel leesprivileges voor het eigen en lagere niveau's. Ze krijgen enkel schrijfprivileges voor het eigen en hogere niveau's.
11. Mensen op hoger niveau zien andere waarden van de data dan mensen op lager niveau.
    Wat zit er in de top-security opslagplaats?
    Laag-niveau: zakdoeken
    Hoog-niveau: kernwapens
    Ja, stom voorbeeld, I know.
12. Statistische bevraging: enkel aggregatie-instructies zijn mogelijk, of resultaten worden benaderend teruggegeven. Query restriction en data swapping om te vermijden dat mensen extra informatie afleiden.
    Query restriction: irritant als je veel queries wil uitvoeren.
    Data swapping: verkeerde waarden.
13. Toestanden voor en na een instructie.
    Bij relationeel: de tuples opslaan die gewijzigd worden.
    Bij OO: een deep copy van de objecten opslaan.
14. Database wordt nogal moeilijk toegankelijk. Dan zou bij elke instructie de hele database gedecodeerd moeten worden. Bovendien moet dan hetzelfde wachtwoord aan iedereen gegeven worden, wat onveilig is.
15. Voordeel: kortere berekeningstijd
    Nadeel: minder veilig
16. Voordeel: eenvoudiger te implementeren
    Nadeel: kost rekentijd voor het dbms.

Hoofdstuk 10

1. Transacties worden gebruikt om te zorgen dat de database niet inconsistent wordt.
   Problemen: gedeeltelijk uitgevoerde instructies, inconsistente data.
2. Omdat een gedeelte van de databank tussen het begin en einde van een transactie inconsistent kan zijn. Als op dat moment een andere transactie wordt uitgevoerd (vb. lezen), kan 'garbage data' gelezen worden.
3. Instructies voor een dbms worden verwerkt als impliciete transacties.
   Expliciete transacties zijn combinaties van instructies die als een logisch geheel moeten worden uitgevoerd.
4. SELECT
   synchronisatiepunt
   UPDATE
   synchronisatiepunt
   UPDATE wordt ongedaan gemaakt
   synchronisatiepunt
   START TRANSACTION...
   UPDATE
   DELETE
   ROLBACK WORK
   synchronisatiepunt en ongedaan maken van UPDATE en DELETE
   SELECT
   INSERT
   ongedaan maken van INSERT
5. SELECT blah from A inner join B

update A, update B
Controleer een of andere voorwaarde: indien voldaan, verander A nog een keer.

Als je voorgaand gedeelte niet als transactie uitvoert, kan een andere instructie de eerste waarde van A gebruiken, terwijl die waarde daarna nog veranderd is.

6. Soft crash: databasebuffers gaan verloren
   Hard crash: volledige/deel van de databank zelf (secundaire geheugen) gaat verloren.
   3 voorbeelden soft crash: stroom valt 3 keer uit
   2 voorbeelden hard crash:
   -harde schijf gooiwedstrijd
   -nucleaire oorlog (maar Killian gaat een bunker zetten!)
7. Crash van de magneetschijf zou zorgen dat de databank verloren gaat.
   Stroompiek kan zowel primaire als secundaire geheugen beschadigen (denk ik).
   Beveiliging tegen stroomuitval kan zorgen dat de databank blijft draaien.
   Ik zou kiezen voor een magneetschijf en beveiliging tegen stroomuitval. Als de magneetschijf beschadigd raakt door stroompieken, is er een back-up magneetschijf en gaan enkel de databasebuffers verloren.
8. Eerst de commit in de log registreren, dan pas uitvoeren.
   Als eerst de commit wordt uitgevoerd, en het systeem crasht voordat het logbestand is aangepast, gaan de buffers (waarin de transactie is uitgevoerd) verloren, en is de transactie verdwenen.
9. Om te zorgen dat niet het logbestand én de database verloren gaan. Je hebt minstens 1 van beide nodig om data te herstellen.
10. Striping en mirroring.
11. Flushing: doorsturen van de databasebuffers naar het secundaire geheugen.
    Nodig: omdat de plaats in het primaire geheugen beperkt is.
12. Steal: flushing mag uitgevoerd worden terwijl er niet-bevestigde transacties zijn.
    No force: flushing mag niet door een impliciete of expliciete commit-instructie.
13. Controlepunten: zorgen dat database en logbestand gesynchroniseerd zijn, zodat men bij soft crashes niet vanaf de vorige back-up moet herstellen.
    Omdat transacties die nog bezig waren tijdens het controlepunt, opnieuw uitgevoerd moeten worden als teruggekeerd wordt naar dat controlepunt.
14. -no steal
    -dus op controlepunten zijn er geen transacties bezig
    Transacties die na het laatste controlepunt zijn uitgevoerd, worden opnieuw uitgevoerd.
15. -steal
    -euhm
16. Alle gegevens worden 2 keer bijgehouden. De schaduwpagina's en werkpagina's hebben bij begin van transactie dezelfde inhoud, en enkel werkpagina wordt tijdens transactie aangepast. Aan het einde van de transactie worden de wijzigingen gekopieerd naar de schaduwpagina's.
    Voordeel: geen logbestand nodig, en undo- en redo-operaties zijn niet meer nodig
    Nadeel: versnippering van het geheugen
17. Ja, men kan de schaduwpagina's op een tweede schijf plaatsen.
    Bij een RAID-systeem kan men met de ene schijf de andere herstellen (als er mirroring is), schaduwpagina's kunnen analoog de database herstellen.

Hoofdstuk 11

1. Parallelle transacties kunnen uitgevoerd worden als er meerdere processors zijn.

Als er echter meer gelijktijdige transacties zijn dan processors, verdelen de processors hun werktijd over de verschillende transacties, zodat ze interleaved verwerkt worden: een gedeelte van een transactie (A) wordt uitgevoerd, vervolgens een gedeelte van een andere transactie (B), en daarna kan weer een deel van A uitgevoerd worden.
Maak hier gewoon maar een tekening voor, da's het handigste.

2. Lost update: transactie A haalt gegevens op, transactie B haalt dezelfde gegevens op. Transactie A schrijft aangepast gegevens weg, transactie B schrijft nog andere aangepaste gegevens weg.
   Gevolg: de weggeschreven gegevens van A zijn nooit gebruikt.
   Voorbeeld: er wordt een gebruiker toegevoegd in de database, en een gebruiker verwijderd. Het aantal gebruikers wordt apart bijgehouden (mag eigenlijk niet: afgeleide data).
   Transactie A leest aantal gebruikers in, transactie B ook.
   Transactie A verhoogt aantal gebruikers met 1 en schrijft weg.
   Transactie B verlaagt aantal gebruikers met 1 en schrijft weg.
   Gevolg: aantal gebruikers is incorrect.
   Uncommitted dependency:
   transactie A schrijft gegevens weg, en transactie B leest die gegevens. Daarna wordt een rollback op transactie A uitgevoerd, waardoor de gegevens die B gebruikt, ongeldig zijn.
   Inconsistente analyse: een transactie past gegevens aan terwijl een andere transactie deze gegevens nog aan het verwerken is.
   Transactie A leest gegevens a in, transactie B past gegevens a en b aan.
   Daarna leest transactie A gegevens b in, en schrijft a+b naar c weg.
3. Dirty read: bij een uncommitted dependency-probleem werkt een transactie verder met gegevens die niet langer geldig zijn. Het lezen van onbevestigde (niet-'gecommitte' data) noemen we 'dirty read'.
   Nonrepeatable read: een transactie vraagt meermaals dezelfde gegevens op, maar krijgt telkens andere waarden (omdat een andere transactie aanpassingen heeft doorgevoerd).
   Phantom read: meermaals een opzoekinstructie uitvoeren, en bij het resultaat een extra record vinden (tussen de 2 opvragingen toegevoegd door een andere transactie).
4. Serialiseren, timestamping, locking, optimistische methodes.
   Mijn dbms? Euh...
5. Serialiseerbaarheid: als instructiesequenties van parallelle transacties om te zetten zijn naar een seriële sequentie, en dezelfde resultaten produceert.
   Als instructies serialiseerbaar zijn, kan men ze zonder problemen gelijktijdig uitvoeren.
6. Isolatieniveau's: instructies worden parallel uitgevoerd indien mogelijk, maar serieel als ze niet geïsoleerd zijn van elkaar. (misschien slecht omschreven)
   Men kan verschillende isolatieniveau's kiezen, afhankelijk van de veiligheid/snelheid die men wenst.
   Ondersteunde niveau's: read uncommitted, read committed, repeatable read, serializable
7. Men kent een timestamp (unieke identificator die de relatieve starttijd van een transactie aanduidt) toe aan iedere transactie.
   Basisregels:
   -als een transactie wil lezen of aanpassen, wordt dat enkel toegestaanals de recentste aanpassing op dat gegeven gebeurde door een oudere transactie.
   -in het andere geval wordt het lezen of aanpassen geweigerd en wordt de transactie ongedaan gemaakt ('rollback') en opnieuw opgestart met een nieuwe, recentere 'timestamp'.
   Uitgebreider met lees-timestamp en aanpassings-timestamp: zie p.324-325
8. De aanpassingsregel van Thomas stelt dat bij stap 2b van het algoritme de aanpassingsoperatie van T veilig kan genegeerd worden zonder T af te breken.
   Dan moet je wel een extra transactie uitvoeren om te zorgen dat de aanpassing van de

recentere transactie later toch nog wordt uitgevoerd. (Wat is dan het nut van de regel??)

9. Een lock is een reservering voor data: een transactie moet een exclusief of een gedeeld lock aanvragen op een gegeven (en dat lock krijgen) voordat het die gegevens kan lezen/aanpassen.
   Basisregels:
   -een transactie die een gegeven wil lezen, moet een gedeelde lock op dat gegeven krijgen van het dbms
   -een transactie die een gegeven wil aanpassen, moet een exclusieve lock krijgen op dat gegeven
   Twee-fasen protocol: transactie kan opgedeeld worden in 2 fasen -- in de eerste fase kan een transactie locks aanvragen en krijgen. In de tweede fase geeft de transacties locks vrij. Tijdens de tweede fase kunnen geen nieuwe locks meer aangevraagd worden.
10. Bij locking-methodes moet minder informatie bijgehouden worden dan bij timestamping (bij timestamping timestamps voor ieder gegeven in de database), en is dus in het algemeen efficiënter. Er is echter het probleem van de deadlocks.
    Locking-methodes zullen dus in veel gevallen snel een resultaat teruggeven, maar in sommige gevallen zal geen resultaat gegeven worden tot het dbms het deadlock opmerkt. Als dus sowieso snelle reactie nodig is, is het beter om te kijken naar timestamping. Als daarentegen in het algemeen een hogere snelheid nodig is, heeft locking een voordeel.

    Men kan ook een keuze maken op basis van welke problemen vermeden worden door de twee verschillende technieken.
11. Granulariteit: men kan timestamps/locks toekennen op het niveau van de volledige database, op een collectie records, op 1 record of zelfs op 1 attribuut van een record.
12. Livelock: als een transactie voortdurend geen lock krijgt (bijvoorbeeld omdat de transactie een exclusief lock nodig heeft, maar er voortdurend andere transacties 'voorsteken' die een gedeeld lock nodig hebben).
    Deadlock: als meerdere transacties geen lock krijgen omdat ze wachten tot de ander locks vrijgeeft.
13. -Met een wacht-voor graaf (veel werk)
    -Met een time-out (valse positieven)
14. Een wacht-voor graaf is een gerichte graaf met knopen die transacties voorstellen en pijlen van transactie A naar B als A wacht om een gegeven te locken dat gelockt is door B.
    Als er cycli voordoen, is er sprake van een deadlock.

    Het dynamisch bijhouden van de wacht-voor graaf zorgt voor een voortdurende kost op de database. Als men een beperkt budget heeft qua hardware, kan het voordelig zijn om slechts nu en dan een wacht-voor graaf op te bouwen (en dan gewoon op 1 moment de database sterk te vertragen).
    Deze sterke vertraging zorgt echter wel dat op dat ene moment gegevens in het geheel niet bereikbaar zijn.
    As usual hangt het af van de omstandigheden, waarvoor de database dient en zo.