Databanken / Relationele gegevensbanken UGent VRAGEN en ANTWOORDEN

geschreven door

indinginf

Relationele gegevensbanken – vragen tijdens interactief uurtje op dinsdag

De typische vragen die hij op het examen stelt, maar niet zoals “wat bedoelt men met …”. Meestal zelf een aantal antwoorden onder een vorm van meerkeuze → motiveer uw antwoord. (Quotering dan gemakkelijker). Op het examen bv. vragen welke uitspraak het meest correct/relevant is.

Vorig jaar zat NoSQL nog niet bij dit vak → kan nu wel op examen komen.

Inhoud

Hoofdstuk 1: Databanksystemen (week 2)

Wat is een databank? --- Vaak op examen ---
Een persistente collectie van gegevens/data. Bedoeld voor de ondersteuning van de operationele activiteiten van een persoon of organisatie.
Persistent: opslag op media dat niet verloren gaat bij het uitschakelen.

Wat zijn de voor- en nadelen van het werken met geïntegreerde en gestructureerde data?
Geïntegreerde data.
Voordeel: geen dubbele data (als gegevens wijzigen moet je het anders overal aanpassen).
Nadeel: bij relationele databanken is data vaak verspreid over meer basisrelaties (tabellen) waardoor bevraging trager werkt

Gestructureerde data:
Voordeel: vaste structuur voor alle tuples (records) wat gemakkelijker te doorzoeken is. Dus snellere en eenvoudigere bevraging.
Nadeel: data moeten vaak worden omgezet naar deze structuur, wat transformatietijd kost. Complexere data-invoer.

Wat zijn databankbuffers?
Geheugenruimte dat gereserveerd is in het primaire geheugen (meestal RAM-geheugen, sneller) en opgedeeld is in pagina's die de grootte hebben van een diskblok. Daardoor past een volledig diskblok in een databankbuffer. De hoeveelheid data die per keer gelezen en geschreven wordt van secundair geheugen (bv. harde schijven) is dan een volledig diskblock. De grootte wordt ingesteld door databaseadministrator. Moet regelmatig naar secundair geheugen geschreven worden om de persistentie te bewaren.

Wat is het nadeel van drielagenarchitectuur?
Een tragere werking van het databankbeheersysteem; elke laag brengt een vorm van vertraging met zich mee. Queries moeten vanuit de externe laag worden omgezet (vertaald), naar corresponderende queries in de conceptuele laag. Queries uit de conceptuele laag moeten worden omgezet naar corresponderende instructies in de interne laag, die dan pas voor uitvoering worden doorgegeven aan het besturingssysteem (voor lezen en schrijven van data op secundair geheugen). Omgekeerd moeten resultaten van instructies worden omgezet naar queryresultaten in de conceptuele laag en deze kunnen op hun beurt nog moeten worden omgezet naar query resultaten in de externe laag. Al deze omzettingen kosten tijd.

Wel voordeel: minder complex voor gebruikers en beter onderhoud van de applicatie.

Waarom zijn gebruikers belangrijk (als component van een databanksysteem)?
Omdat het databankbeheersysteem om beveiligingsredenen expliciet onderscheid moet kunnen maken tussen verschillende gebruikers (rechten). Verschillende gebruikers kunnen verschillende privileges krijgen om taken uit te voeren op een gegeven databank. Ook in licht van privacy, GDPR, wettelijke voorschriften die zeggen dat je geen data mag opslaan als de identiteit van die persoon eruit af te leiden is.

Heb je altijd een database nodig, of kan je ook naar bijvoorbeeld een bestand schrijven? Dat kan, het zal ook snel toegankelijk zijn, maar functies zoals beveiliging, meerdere gebruikers tegelijk, … zou je dan zelf moeten toevoegen.

Waarom maakt men binnen een onderneming best onderscheid tussen data-administrators en databankadministrators? Het zijn verschillende profielen met verschillende verantwoordelijkheden, waarvoor verschillende competenties vereist zijn: data-administrator = managementfunctie (bv. houdt formaten bij); databaseadministrator = technische functie.

Diegene die verantwoordelijk is voor de inhoud van een databank (data-administrator) is best niet diegene die verantwoordelijk is voor de goede werking van het databanksysteem (databaseadministrator). Management zorgt voor wat er moet worden opgeslagen en wie wat mag doen op de databank.

Bij ingrijpende veranderingen zal DA zeggen wat waar moet veranderd worden (bv. Frank → euro).

Als één rol, zou DBA te veel macht hebben, vraagt ook 24/7 bijstand (vaak team). Om veiligheidsredenen geef je de DBA best niet alle verantwoordelijkheid over de databank.

Hoofdstuk 2: databankmodellen (week 3)

[Niet van prof] Wat is een virtuele ouder-kind relatie? (aangeduid met stippellijnpijlen).
Waarom zijn deze in de meeste hiërarchische databanksystemen beperkt tot 1?
Boomstructuurtype, ten hoogste 1 ouder. Maar het kan ook handig zijn om aan andere entiteiten te kunnen → virtuele ouder-kindrelatie. Relatie tussen ouder en kind moet 1-N relaties zijn, (ook de virtuele ouder-kindrelaties.) Bv. zowel 1 eigenaar als 1 schilder per schilderij, wel meerdere schilderijen mogelijk bij beide. Je kan wel meerdere virtuele ouders definiëren, maar binnen 1 ouder-kind relatie is het 1-N.

[Niet van prof] Wat zijn de voor- en nadelen van een hiërarchisch databankmodel en een netwerkdatabankmodel? Wanneer kies je voor de ene en wanneer voor de ander?
Een netwerkdatabank laat het toe om veel soorten queries heel efficiënt uit te voeren (veelvoud aan pointerlijsten waarmee records aan elkaar gelinkt worden). Aanpassingen, verwijderingen en toevoegingen zijn trager omdat doorgaans de pointerlijsten mee aangepast moeten worden wat extra tijd vraagt.
Een hiërarchische databank werkt enkel efficiënt voor bevragingen waarbij men de hiërarchische links tussen de records kan gebruiken. Aanpassingen, verwijderingen en toevoegingen vragen minder tijd omdat er minder aanpassingen moeten gebeuren aan de dataopslag (hiërarchische sequenties).

Vb.: netwerkdatabankmodel voor databanken die veel bevraagd worden en zelden aangepast worden.

[Niet van prof] Wat zijn atomaire waarden en hebben deze voor- en nadelen?
Domeinwaarde wordt als één atomair geheel beschouwd, één waarde, geen verschillende onderdelen om te behandelen. Bv. Mona Lisa bestaat uit twee woorden, is het atomair? Ja, want volgens het databankmodel en het databankbeheersysteem is het één waarde, één karakterstring. Dus meerdere waarden / tabellen mogen niet.
Voordeel: het houdt databankmodel eenvoudig. Maar ook nadeel, want je kan geen complexe datatypes gebruiken.

Operationele modellen

Wat betekent het als je de data gaat modelleren in functie van efficiënt bewerken en vragen?
Men gaat de databank opbouwen met de bedoeling om de belangrijkste bevragingen en datamanipulatie-operatoren zo efficiënt mogelijk te kunnen uitvoeren. Dus in functie van applicatie.

Voordeel: snellere bewerkingen op de databank (zoeken, toevoegen, verwijderen of aanpassen)
Nadeel: de databank is sterk gekoppeld aan een (bepaald type van) toepassing. Andere toepassingen zullen veel trager werken.

Hiërarchisch databankmodel

Wat wil logische nabijheid zeggen? Wat is fysieke nabijheid? Wat is het voordeel van beide?
Logische nabijheid:
Records moeten logisch gezien worden opgeslagen volgens een recordsequentie. Daardoor staan kindrecords logisch gezien dicht bij hun ouderrecord.

Fysieke nabijheid:
Dit heeft betrekking op de plaats waar een record effectief wordt opgeslagen op secundair geheugen. Het streefdoel is om logische nabijheid zo goed als mogelijk om te zetten naar fysieke nabijheid (2 records die logisch gezien naast elkaar staan zouden dus ook fysiek naast elkaar geschreven worden op de harde schijf). Maar de volgorde van toevoeging van kindrecords en eventuele aanpassingen kunnen ervoor zorgen dat fysieke nabijheid (dit is in hetzelfde diskblok) niet mogelijk is (omdat er geen plaats meer is in dat blok). In dat geval moet er naar een ander diskblok geschreven worden wat ervoor zorgt dat de records niet meer met dezelfde leesoperatie kunnen gelezen worden (per leesoperatie wordt 1 diskblok gelezen). Als je bv. een string vervangt door een grotere string en het ergens anders moet plaatsen → slechtere fysieke nabijheid.

Voordeel:
- records met een fysieke nabijheid kunnen heel snel samen opgevraagd worden
- bij records met een logische nabijheid is dat niet altijd het geval, maar de meeste records zullen ook fysiek dicht bij elkaar staan zodat het lezen nog altijd efficiënter zal zijn dan in het geval waar records in willekeurige diskblokken worden weggeschreven.

Nadeel: vroeg of laat krijg je versnippering van geheugen, zodat je diskreorganisatie nodig hebt (kan lang duren en tijdens is de data tijdelijk niet toegankelijk.)

Structurele modellen

Wat is het nadeel van een relationele databank en welke voordelen heeft het t.o.v. operationele databankmodellen?

Nadeel:
- een relationeel databankschema bevat doorgaans veel basisrelaties (tabellen) (data verspreid) waardoor er bij bevragingen veel trage join-operaties zullen moeten worden uitgevoerd.
- basisrelaties zijn heel eenvoudig opgebouwd waardoor het vaak moeilijk is om complexe informatie om te zetten naar tabelformaat. Informatieverlies (impedance mismatch) mogelijk door vertaling. Enkel geschikt voor informatie die je makkelijk in tabelstructuur kan gieten.

Voordelen:
- gestandaardiseerd (je bent niet gebonden aan ontwikkelaar van het databanksysteem)
- complexere bevraging is mogelijk --via SQL-- (maar kan traag zijn)
- een databankschema is een abstractie is van de realiteit waarover men informatie wil bijhouden; dit schema is niet afgestemd op een bepaald (type van) applicatie
- wiskundig onderbouwde bevragingstaal; daardoor kan de bevraging volledig afgehandeld worden door het databankbeheersysteem en hoeft de gebruiker niet te programmeren. Een SQL-query doorgeven volstaat.

Voorbeeld van gegevensstructuur die heel moeilijk kan gemodelleerd worden in een relationeel databankschema? (Machine learning). Data die sterk gelinkt is bij elkaar (zoals gelinkte lijsten), zoals familiestamboom, bevragen zeer moeilijk. Ook niet voor versies (zoals wel bij datawarehouses). Gelinkte lijsten, boomstructuren, graafstructuren,...
Dergelijke datastructuren modelleren in een basisrelatie gaat, maar de bevraging vraagt het recursief doorzoeken van deze basisrelatie. Iets wat heel traag en inefficiënt werkt.

Semantische modellen

Waarom is puur objectdatabankmodel technisch beter dan het objectrelationeel databankmodel?
Bij een puur objectdatabankmodel worden de objecten opgeslagen als objecten en kan men gemakkelijk via links navigeren van een object naar een gekoppeld, gerelateerd object. Bij een objectrelationeel databank model moeten de objecten nog steeds vertaald en omgezet worden naar tabelformaat. Dit is minder efficiënt en leidt doorgaans ook tot een complexere, tragere bevraging.

Nadelen:
- niet gestandaardiseerd → hoger risico op problemen als de leverancier van de markt zou verdwijnen.
- complexer om mee te werken (men moet kunnen programmeren) / hogere leercurve
- meestal gekoppeld aan een beperkt aantal OO-programmeertalen.

Databankmodellen

Welk databankmodel zou je gebruiken om een familiestamboom te modelleren?
Hiërarchisch databankmodel of graafdatabankmodel (zie later bij NoSQL systemen).
Een netwerkdatabankmodel of puur objectdatabankmodel kan ook werken, maar zeker geen relationeel databankmodel.

NoSQL-databanken

Wat bedoelt men met een databank zonder vast databankschema?
Een databank waarbij men niet werkt met een vast databankschema.
Hierdoor moeten data niet worden omgezet naar een vaste structuur en verloopt het toevoegen van data heel vlot. Let wel: de data zelf zijn doorgaans wel nog gestructureerd, maar elk record kan zijn eigen structuur (‘recordtypestructuur’) hebben. Zie hoofdstuk rond NoSQL databanken.

Hoofdstuk 3: Conceptueel ontwerp (week 4)

Conceptueel ontwerpproces

Abstraheren, afbakenen en afwerken.
Abstraheren: focus op relevante, gemeenschappelijke karakteristieken van entiteiten die je moet modelleren en opnemen in de databank. Alle aspecten waarvoor data moet worden opgenomen, schetsen en visueel opnemen in EER-diagram.

Afbakenen: je houden aan de opdracht en het niet moeilijker gaan maken dan nodig is.

Afwerken: zorg ervoor dat alle informatie die moet gegeven worden ook gegeven wordt (cardinaliteiten en participatiegraden bij relatietypes, sleutels, ...). Aandacht voor details.

Waarom dient conceptueel ontwerp afhankelijk te gebeuren van enig databankmodel?
De keuze van het databankmodel (gereedschap) moet in functie zijn van de verwachtingen (opdracht/probleem) -- niet omgekeerd. Gebruik het databankmodel dat het beste pas om de data te structureren en verwerken volgens de noden van de gebruikers.
Laat je bij het conceptueel ontwerp niet leiden of beperken door een vooringenomen keuze van datastructuren. Je gaat misschien wel later moeten terugkoppelen.
Tegenwoordig wordt het soms complex: misschien meerdere implementaties: diagram opsplitsen.

EER-modellering

EER-modellering ondersteunt amper modellering van functionaliteit. Waarom is dit een beperking? Hoe kan je hieraan verhelpen? Kan een modelleringstechniek zoals UML hierbij helpen en EER-modellering vervangen? Data worden gekarakteriseerd door structuur en gedrag. Naast de datastructuur is het belangrijk om ook te kunnen meegeven welke speciale functionaliteiten van toepassing zijn (in functie van databankbeheer). Voorbeelden zijn integriteitscontroles, afgeleide data berekenen, ...

Verhelpen: relevante functionaliteiten opsommen zodat je ze kan meenemen in de andere fasen van databankontwerp.

UML is speciaal bedoeld voor objectgeoriënteerde modellering. Als je deze techniek gebruikt, word je in de richting van objectgeoriënteerde databanktechnologie geduwd, terwijl conceptueel ontwerp databankmodelonafhankelijk moet zijn (zie ook vorige vraag).

Welke problemen zie je bij het gebruik van een casetool voor EER-modellering?
Case-tools zijn vaak constructeur-(databanksysteem)-gebonden. Ze gebruiken geen uniforme notaties (dus niet altijd gestandaardiseerd). Ze zijn vaak beperkt in hun mogelijkheden en ondersteunen niet alle faciliteiten van EER-modellering.

In de praktijk kan het gebruik van een case-tool zeer nuttig zijn, maar niet voor educatieve doeleinden.

Aandachtspunten bij het bouwen van een EER-diagram

Wanneer kies je voor een samengesteld attribuut en wanneer voor een entiteitstype? (Soms een moeilijke afweging.) Een entiteitstype beschrijft een entiteit. Een samengesteld attribuut beschrijft een kenmerk van een entiteit.

Hint: denk aan een relationele databank. Een entiteitstype zal hierin doorgaans gemodelleerd worden als een basisrelatie ('tabel'). Een samengesteld attribuut als attributen van een basisrelatie (één voor elke basiscomponent).

De vraag komt in deze context dus neer op kiezen voor een extra basisrelatie of kiezen voor extra attributen in een bestaande basisrelatie. (Hoe meer basisrelaties hoe inefficiënter de bevraging achteraf, maar soms kan je niet anders -- bv. je kan geen relatietype definiëren voor een samengesteld attribuut). Dus als er relaties nodig zijn met andere entiteiten → entiteitstype.
Bijvoorbeeld geen datum (dag, maand, jaar) in een aparte tabel steken![2]

Waarom moeten samengestelde sleutels gemodelleerd worden via een samengesteld attribuut? Omdat je anders bv. geen onderscheid kan maken tussen de situatie waarbij een entiteitstype twee enkelvoudige sleutels heeft en de situatie waarbij het entiteitstype één samengestelde sleutel heeft die is samengesteld uit twee componenten.

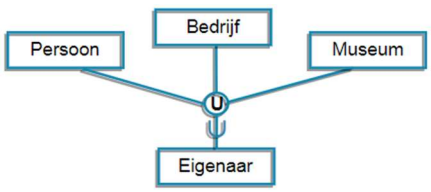
Hoe dien je een "deelverzameling van"-symbool te interpreteren bij overerving? Als hij enkel uniek is binnen de context van een ander entiteitstype. (Niet echt "als het niet op zichzelf kan bestaan").
De verzameling van alle entiteiten van een subtype is een deelverzameling van de verzamelingen van alle entiteiten van het supertype van dat subtype.

Wanneer kies je in de praktijk voor een zwak entiteitstype? Enkel wanneer het nodig is om de identiteit van een bepaalde entiteit te laten afhangen van een ander entiteitstype.
Alleen wanneer de sleutel van een entiteitstype (voor een deel) bepaald wordt door de sleutel van andere entitiiteitstypes. In dat geval modelleer je het eerste entiteitstype als zwak entiteitstype.

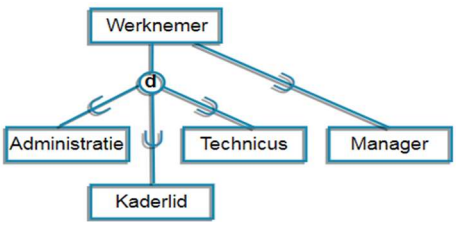
Voorbeeld: stel dat elk patiëntendossier een patiëntnummer moet bevatten. Dan modelleer je patiëntendossier als zwak entiteitstype met identificerend regulier entiteitstype patiënt. Elk dossier kan dan bv. nog een uniek volgnummer hebben maar zal geïdentificeerd worden door een samengestelde sleutel (patiëntnummer, volgnummer).

Wat is het verschil tussen meerwaardige overerving (generalisatie) en een categorie. **Examen:** categorieën zullen niet in ontwerpdeel/oefeningen gevraagd, maar kan wel op theorie. Bv. deze vraag.

In beide gevallen heb je een subtype, eigenaar, en heb je meerdere supertypes.



categorie. Eén symbool (U)



Alle managers zijn werknemers.

Een categorie is een subtype.

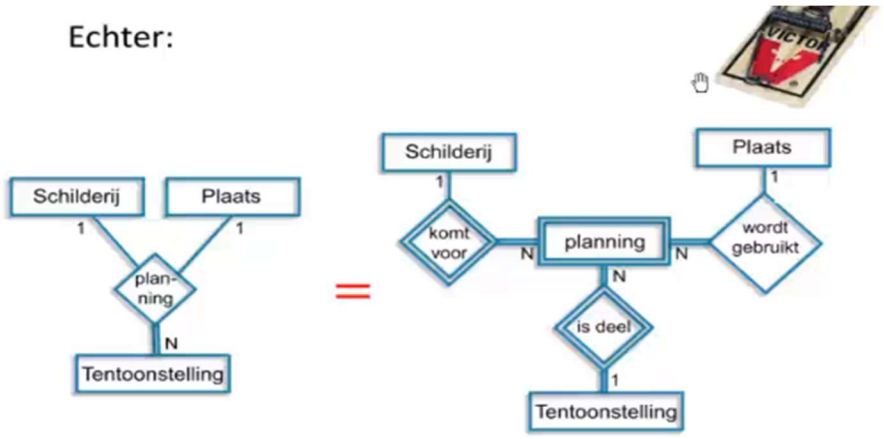
Bij een meerwaardige overerving heeft een subtype meerdere supertypes en erft het alle karakteristieken (attributen en relatietypes) van elk van deze supertypes.

Een categorie heeft ook meerdere supertypes maar erft enkel de karakteristieken van **juist één** van deze supertypes.

Mag je M-N relatietypes modelleren via een apart entiteitstype?
Vanuit conceptueel standpunt gezien is het geen goede praktijk om M-N relatietypes te modelleren via een apart entiteitstype. Je "verliest" dan immers de informatie dat het gaat over een M-N relatietype. Een ervaren databankontwerpen kan dit wel nog achterhalen maar je EER-diagram wordt er minder duidelijk door.

Echter in de praktijk kunnen EER-diagrammen ook gegenereerd worden via reverse engineering, wat wil zeggen dat men van een relationeel databankschema vertrekt om via een case-tool het onderliggend EER-diagram te reconstrueren.
Zoals we later zullen zien bij logisch ontwerp, zullen M-N relatietypes bij de omzetting van een EER-diagram naar een relationeel databankschema omgezet worden naar een apart entiteitstype. Daardoor zullen in het resulterend EER-diagram na reverse engineering meestal extra entiteitstypes om M-N relatietypes te modelleren. Dit is geen fout, maar draagt evenmin bij tot de duidelijkheid.

Wat is het belang van ternaire relatietypes? Strikt genomen geen ternaire relatietypes nodig. Maar je kan een ternair relatietype modelleren adhv een extra zwak entiteitstype en dan met binaire relatietypes, maar het is een valkuil. Het wordt aangeraden om het niet te gebruiken omdat er vaak fouten bij gemaakt worden. Daarom beter zwak entiteitstype.



Note: in rechtertekening kan je ook de uiterst rechtse ruit dubbel tekenen i.p.v. linkse.

Ternaire relatietypes leggen verwantschappen vast tussen entiteiten van drie entiteitstypes. Een ternair relatietype kan je niet vervangen door drie binaire relatietypes tenzij je een extra (zwak) entiteitstype invoert. (Zo moet je te werk gaan als je case-tool geen ternaire relatietypes ondersteunt.)

Pas op voor de 'connection'-trap (zie slide).

Hoofdstuk 4: Relationeel databankmodel (week 5)

Ook enkele oud-examenvragen in deze sessie.

[Niet van prof] Wanneer opteer je best om een cartesiaans product te gebruiken aangezien dit vaak vermeden wordt?
In de praktijk kan je cartesiaans product moeilijk vermijden aangezien het aan de basis ligt van een join-operatie. Cartesiaans product is een zwak punt.

[Niet van prof] Is het best om cyclische referenties te vermijden?
Cyclische referenties vermijden? Nee, cyclische referenties moet je implementeren als je ze nodig hebt, je kan er niet aan uit. Je kan ze niet vermijden als je ze nodig hebt.

Structurele aspecten

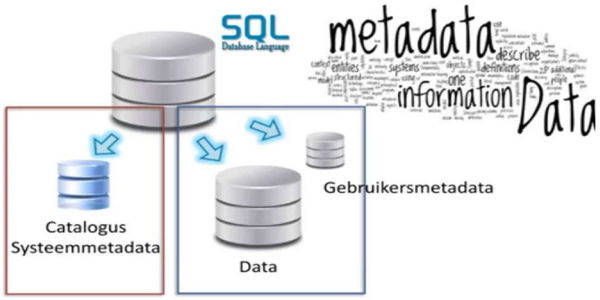
Waarom spreekt men in het relationeel databankmodel van basisrelaties en niet van tabellen?
Een tabel is maar een visuele voorstelling van een basisrelatie (of afgeleide relatie). Men wil het onderscheid maken om een concept te hebben binnen het databankmodel om te beschrijven wat je wil gaan doen. Een basisrelatie is als concept gekozen boven relatie, omdat relatie echt een wiskundig concept is, een basisrelatie is iets meer.

Het is belangrijk om een onderscheid te maken tussen:
- *basisrelatie*: het basisconcept uit het relationeel databankmodel
- *tabel*: een visualisatie van een basisrelatie; een tabel suggereert een ordening van de tuples en een ordening van de attributen van een basisrelatie, terwijl deze volgens de definitie van een basisrelatie (verzamelingen) niet geordend zijn. Operatoren uit de relationele algebra maken ook geen gebruik van deze ordeningen.
- *wiskundig concept relatie*: de extensie van een basisrelatie kan je zien als een wiskundige relatie (zie uitleg boek/slides) maar een basisrelatie heeft ook een relatieschema wat je niet hebt in een wiskundige relatie, vandaar de aparte benaming voor de duidelijkheid.

Wat is het belang van atomaire datatypes? Twee redenen om met atomaire datatypes te werken.
1) Complexiteit datamodel beperken, zo eenvoudig mogelijk (vanuit standpunt dbms). Basisrelaties zijn eenvoudig opgebouwd. Deze eenvoud draagt bij tot het succes van het relationeel databankmodel, maar is tegelijkertijd ook een zwak punt (zie verder in lessen rond objectrelationele databanken).
2) Efficiënt opslaan van gegevens

Waarom mag een catalogus enkel worden gelezen? Wat is het verschil tussen de metadata uit een catalogus en gebruikersmetadata? --- Examenvraag vorig jaar --- (wat zou het probleem zijn als je een catalogus zou kunnen aanpassen)
a) Een catalogus wordt door het databankbeheersysteem aangemaakt en bijgehouden. Het dbms maakt gebruik van de catalogus om haar werking te sturen. Als gebruikers catalogusdata zouden aanpassen dan kan dat de catalogus corrupt maken waardoor de goede werking van het dbms in het gedrang komt.

b) Traditioneel werden data uit de catalogus de metadata van een databank genoemd. Met de komst van multimedia kreeg 'metadata' een ruimere betekenis en kan dit ook duiden op data over de data uit een databank die niet in de catalogus zijn opgenomen. Deze data noemen we gebruikersmetadata.



Waarom is het belangrijk dat views niet gematerialiseerd zijn?
Materialiseren = view eenmalig uitvoeren en queryresultaat wegschrijven in databank.

Gematerialiseerde views zouden overtollige dataopslag betekenen, wat risico's met zich meebrengt voor data-inconsistentie. Afgeleide data opslaan in een databank is eveneens sterk af te raden omwille van risico's op inconsistentie. Dus: redundantie voorkomen.

Een view is een externe kijk op een databank (cf. externe laag van de drielagenarchitectuur van een databankbeheersysteem). Dus geen kopie van een deel van de databank.

(Bij een datawarehouse zullen views meestal wel gematerialiseerd zijn, historiek belangrijk daar.)

Waarom wordt het onderscheid tussen basisrelaties en afgeleide relaties naar gebruikers toe transparant gehouden? Transparantie kan niet volledig verborgen blijven, bv. als hij wil toevoegen en het lukt niet, weet hij dat er niet aan de 3 voorwaarden (*) voldaan is.

*
- 'With check option'
- Niet conflicterend met definiërende expressie
- Geen manipulaties op afgeleide data

a) Omwille van beveiligingsredenen is het belangrijk dat een gebruiker geen verschil ziet tussen basisrelaties en afgeleide relaties: als de gebruiker in de waan is van alle data te zien gaat hij/zij ook niet op zoek gaan naar verborgen data.
b) view = 3e laag van de 3 lagen-architectuur, query's rechtstreeks op basisrelaties.
Door geen onderscheid te maken wordt de SQL-taal niet overlaten met extra complexiteit. Dit maakt het opstellen van SQL queries eenvoudiger.

Hoe kan je binnen het relationeel databankmodel vermijden dat je nullwaarden of defaultwaarden nodig hebt voor het behandelen van ontbrekende informatie? --- examen vorig jaar --- +- min 31:00
Je kan dat vermijden door extra basisrelaties in te voeren en daarbij onderscheid te maken tussen entiteiten waarvoor data gekend zijn en entiteiten waarvoor data ontbreken.
Nadeel: Hierdoor introduceer je 'artificiële' complexiteit, krijg je meer basisrelaties in je databankschema en werkt de bevraging doorgaans minder efficiënt (complexere queries).
Er zijn auteurs/wetenschappers (Chris Date) die aanraden om zo te werken. Prof zelf voorstander van het gebruik van nullwaarden.
Je zou ook een onzekerheidsindicatie (probabliteitsdistributie) kunnen meegeven.

Hoe kan je foutieve queryresultaten voorkomen wanneer je werkt met nullwaarden?
---examen vorig jaar--
In de WHERE-clausule meegeven dat de records met een null waarde moeten weergegeven worden.
M.a.w. in de unie nog een unie toevoegen select artiestID where Artiest where geboren IS NULL.

```
SELECT A_ID                    SELECT A_ID
FROM Artiest          UNION    FROM Artiest
WHERE Geboren >= 1800          WHERE Geboren < 1800
```

Door expliciet te bevragen naar tuples met een Null-waarde en deze mee te nemen in het resultaat van de query. Dat kan door een 'IS NULL'-conditie mee te geven in the WHERE-clause van een query.

Integriteitsaspecten

Waarom zijn alternatieve sleutels belangrijk in de praktijk? --examenvraag 2de zittijd vorig jaar---
In het kort: om uniciteit af te dwingen.
Een alternatieve sleutel zorgt ervoor dat elk tuple van de basisrelatie een unieke waardencombinatie moet hebben voor de attributen die deel uitmaken van de alternatieve sleutel.

Als je weet dat attributen een unieke waarde moeten hebben en deze attributen vormen geen primaire sleutel, dan is het omwille van de dataintegriteit toch belangrijk om die uniciteit af te dwingen in de databank. Dat doe je dan met een alternatieve sleutel (UNIQUE specificatie).

Hoe zie je zelf de balans tussen het gebruik van stored procedures en het coderen van businesslogica in de toepassingsprogramma's? Wat zijn de voor- en nadelen van stored procedures? (Niet over triggers). +- keuze maken voor het ene of het andere.

Complexere voorwaarden voor dataintegriteit (validatielogica) worden best geïmplementeerd via stored procedures. "Andere businesslogica zou ik niet te veel implementeren via stored procedures."
Voordelen:
- Centraal beheer. Alle applicaties maken gebruik van dezelfde centrale code.
- Gemakkelijk te onderhouden; aanpassingen gebeuren op één plaats.

Nadelen:
-Tragere verwerking door extra communicatie met het databankbeheersysteem (stored procedure-call)
- mogelijke overbelasting van het databankbeheersysteem. Maakt het toepassingsprogramma ook meer afhankelijk van het databankbeheersysteem.

Enkel de zaken die betrekking hebben op data en de database op zich, in stored procedures schrijven. Dingen die betrekking hebben op programma en niet op data in toepassingsprogramma houden.

Gedragsaspecten

Vanwaar komt de voorwaarde dat relatieschema's van hetzelfde type moeten zijn bij unie-, doorsnede- en verschiloperaties? Als je te maken krijgt met basisrelaties met verschillend schema, zit je met een probleem. Je kan geen appelen met peren, de unie van nemen.

Dit is voor de eenvoud van de query-verwerking. Enkel (verzamelingen van) tuples met een compatibele structuur kunnen worden bewerkt via unie, doorsnede en verschil.

Mocht men deze voorwaarde laten vallen moet men een oplossing vinden voor de attributen die niet gemeenschappelijk zijn in beide relatieschema's. Dit zou dan aanleiding geven tot nullwaarden in de resultaten van de operaties.

PS: er bestaat een OUTER UNION operator die hiervoor zorgt, maar deze behoort niet tot de basisoperatoren van de relationele algebra en valt buiten het bestek van de cursus.

Waarom vormt een Cartesisch productoperatie een bottleneck? Kan je een voorbeeld geven waarbij je bij een join-operatie gebruikt maakt van een "verschillend van"-operator i.p.v. een gelijkheidsoperator?

De Cartesisch productoperatie van twee basisrelaties combineert elk tuple van de ene basisrelatie met elk tuple van de andere basisrelatie. (Bv twee basisrelaties met resp. 1.000.000 en 10 tuples resulteren in een basisrelatie met 10.000.000 tuples.) Hierdoor krijgt men vaak grote basisrelaties als tussenresultaat en deze moeten ook in RAM-geheugen worden bewaard en verder verwerkt.

Voorbeeld: met een query op een basisrelatie 'student' duo's moet vormen van twee studenten (voor bv. groepsindeling bij projectwerk), kan je dat doen door bv. op het resultaat van het Cartesisch product van student met zichzelf de join voorwaarde op te leggen dat de studentid's (die fungeren als primaire sleutelwaarde en vreemde sleutelwaarde) verschillend moeten zijn. Je moet dan wel nog ontdubbelen van (stud_A,stud_B) en (stud_B,stud_A) gaan voorkomen.

Wat is het praktisch belang/voordeel van het gesloten zijn van relationele algebra?

Gesloten: elke operator werkt in op basisrelaties, resultaat is weer een basisrelatie. Waardoor de resulterende basisrelatie weer gebruikt kan worden → je kan algebraïsche expressies bouwen.

Hierdoor zit ook alle functionaliteit die nodig is om een query op te lossen vervat in de algebraïsche expressie, waardoor het databankbeheersysteem de query volledig zelf kan oplossen. Dit voorkomt dat een gebruiker nog moet gaan programmeren om het resultaat van een query op te bouwen en te verwerken, wat het geval is bij andere databankmodellen. M.a.w. je kan rechtstreeks SQL queries doorgeven aan het dbms zonder dat je daarbij nog hoeft te programmeren.

Hoofdstuk 5: Logisch ontwerp (week 6)

EER-mapping zal vooral ondervraagd worden door mapping, mogelijks ook bij examenoefeningen erover. Maar niet echt theoretische vragen daarover.

EER-mapping

Welke overwegingen zijn belangrijk bij het kiezen van een keuzemogelijkheid bij de mapping van een specialisatie/generalisatie?
Het is voordelig om het aantal basisrelaties in relationeel databankschema te beperken. Hoe meer basisrelaties, hoe meer tijdrovende join-operaties kunnen nodig zijn bij het verwerken van de SQL queries.

Een basisrelatie van een subtype wegwerken kan enkel als het corresponderend entiteitstype niet apart betrokken is in relatietypes. Als dit wel zo zou zijn riskeer je informatieverlies te hebben (nl. dat enkel entiteiten van dat subtype in een relatie van het relatietype kunnen voorkomen).

Een basisrelatie van een supertype wegwerken, impliceert dat alle attributen en relatietypes van het supertype moeten worden overgezet naar elk van de subtypes. Dit kan ook een impact hebben op de complexiteit van bepaalde queries.
Beschouw het wegwerken van basisrelaties echt als een vorm van optimalisatie die je enkel doorvoert als de modellering erdoor vereenvoudigd wordt en dit geen informatieverlies veroorzaakt.

Welke overwegingen zijn belangrijk bij een keuzemogelijkheid bij de mapping van binaire 1:1 en binaire 1:N relatietypes? (3 mogelijkheden: samenvoegen, beide basisrelaties behouden, aparte basisrelatie toevoegen).

Algemeen is het af te raden om een extra basisrelatie in te voeren voor het modelleren van een 1:1 en 1:N relatietype. Uitzonderingen zijn waar het relatietype zelf veel attributen zou hebben. (Meer basisrelaties in een databankschema impliceert dat er doorgaans meer join-operaties nodig zullen zijn in de queries).

De optie om bij 1:1 relatietypes een basisrelatie weg te werken (waarvan het corresponderend entiteitstype totaal participeert) is enkel is enkel zinvol als dit geen aanleiding geeft tot informatieverlies (bv. omdat het betreffende entiteitstype participeert in andere relatietypes).

Normalisatie

Op het examen moet je niet kunnen normaliseren. Is voor geavanceerde databanken.
Feit: normalisatie is ontworpen voor relationele databankmodellen.

Kan je normalisatie gebruiken voor relationeel databankontwerp?
Theoretisch ja, maar dan moet je alle attributen waarvoor je informatie wil opnemen in je databank samennemen in één grote basisrelatie en deze normaliseren.

In de praktijk is het vaak een heel moeilijke aanpak omdat databankschema's vaak veel basisrelaties bevatten die op hun beurt weer veel attributen kunnen bevatten. Verder is het veel moeilijker (zeker voor iemand met weinig ervaring) om een functioneel afhankelijkheidsdiagram op te bouwen, dan om een EER-diagram op te bouwen.

Vandaar: gebruik normalisatie voor kwaliteitscontrole van een relatieschema en EER-modellering voor databankontwerp. Normalisatie als kwaliteitscontroletechniek, niet als ontwerptechniek.

Kan je normalisatie gebruiken als kwaliteitscontroletechniek bij andere databankmodellen?
Prof: Normalisatie zorgt ervoor dat het samen gebruiken van attributen binnen dezelfde basisrelatie zo weinig mogelijk aanleiding geeft tot overtollige dataopslag, dataverlies en update-anomalieën. Normalisatie is specifiek ontworpen voor het relationeel databankmodel.

Echter functionele afhankelijkheden zijn eigen aan de data, niet aan het databankmodel. Dus kan men in principe normalisatieprincipes ook gaan toepassen om problemen te vermijden bij het samennemen van attributen in hetzelfde recordtype of in dezelfde klassendefinitie.
Hoewel bij mijn weten nog niet voorgesteld in de literatuur zou men dus ook in andere databankmodellen een vorm van normaliseren kunnen gaan toepassen.

Kan je normaliseren als je geen semantische kennis hebt over de opbouw van basisrelaties?
Neen. Het vinden van irreducibele functionele afhankelijkheden tussen deelverzamelingen van attributen vormt de grondslag van het normaliseren.
Deze afhankelijkheden zijn eigen aan de data en kan je enkel maar te weten komen door de data goed te begrijpen, dit is de betekenis (semantiek) van de data goed te kennen.

Waarom wordt de eerste normaalvorm ingevoerd? Welke consequenties zijn hieraan verbonden?
De eerste normaalvorm werd ingevoerd om te controleren of een basisrelatie wel voldoet aan de definitie van een basisrelatie. Men focust daarvoor op de atomiciteit van de gebruikte datatypes.
Als we later gebruik maken van het objectrelationeel databankmodel zijn ook niet-atomaire datatypes toegestaan en moet de voorwaarde van de eerste normaalvorm worden afgezwakt (zie later bij het hoofdstuk rond objecttechnologie). Doe je dit niet dan wordt bij het normaliseren het niet-atomair datatype omgezet naar een atomair datatype in een andere basisrelatie (zie voorbeeld van Verblijfplaats: varchar array[5]), wat het nut van niet-atomaire datatypes teniet zou doen.

Onder welke omstandigheden zou je denormaliseren?
Nadeel: mogelijks consistentieproblemen door overtollige data.

Denormaliseren komt neer op het afstemmen van je databankschema op het efficiënt uitvoeren van bepaalde queries. Dit is eigen aan operationele databankmodellen en expliciet niet het uitgangspunt bij structurele databankmodellen (in het bijzonder het relationeel databankmodel).

De applicatie kan echter vereisen dat er gedenormaliseerd wordt gewerkt omdat anders bepaalde queries niet tijdig kunnen worden verwerkt. In zo een geval kan je niet rond denormaliseren.
De databaseadministrator dient dan maatregelen te nemen om update-anomalieën zo goed als mogelijk te vermijden. (Door bv extra controles en aanpassingen via triggers of door het gebruik van de databank strak onder controle te houden -- enkel gebruikers die weten wat ze doen, mogen de databank aanpassen).

Hoe gaan de technieken voor EER-mapping en normalisatie samen?
EER-mapping is een methode om een databank te ontwerpen. Als je EER-modellering goed toepast, is het resulterende databankschema normaal gezien genormaliseerd tot 6NF (waardoor extra normaliseren overbodig is).
Normalisatie is een kwaliteitscontrole techniek voor relatieschema's uit een databankschema. Dat gebruik je wanneer je een bestaande databank wil analyseren, aanpassen of uitbreiden.
Bij twijfel over EER-mapping, kan normalisatie nuttig zijn als extra controletechniek voor een beperkt aantal basisrelaties (waarover er twijfel heerst).

Hoofdstuk 6: Fysiek ontwerp (week 7)

FYI: Aanpassen van databankschema is geen datamanipulatie!

Reverse engineering

Wat gebeurt er met M:N relatietypes wanneer je eerst forward engineering en nadien reverse engineering toepast?

Forward engineering:

Bij het opbouwen van een relationeel databankschema wordt elk M:N relatietype omgezet naar een basisrelatie.

Reverse engineering:

Bij het actualiseren van het EER-diagram wordt elke basisrelatie door de meeste case-tools omgezet naar een entiteitstype.

Hierdoor zullen er geen M:N relatietypes meer voorkomen in het geactualiseerde EER-diagram en verlies je semantische informatie: een M:N relatietype wordt nu vervangen door een extra entiteitstype en twee 1:N relatietypes. Zolang je blijft werken met het relationeel databankmodel is dit geen probleem, maar als je het EER-diagram zou gebruiken om een databankschema volgens een ander databankmodel op te bouwen moet je opletten.

SQL

Wat zijn de voor- en nadelen van een SQL-dialect?

Voordeel: meer mogelijkheden/functies, meer flexibiliteit, groter gebruiksgemak, extra datatypes, …

Nadeel: : wanneer men deze faciliteiten gebruikt is men gebonden aan het databanksysteem dat het dialect gebruikt. Omschakelen naar een ander databanksysteem vereist dan dat concepten die eigen zijn aan het dialect gepast worden omgezet naar de SQL-standaard of het dialect van het nieuwe databanksysteem.

Als men zich beperkt tot de SQL-standaard heeft men deze omzettingsproblemen niet en is het onderhoud van toepassingsprogramma's eenvoudiger.

Waarom moet je voorzichtig zijn met de Cascade-optie in DDL-instructies?

Het gebruik van 'Cascade' bij het verwijderen van een databankcomponent heeft als neveneffect dat alle databankcomponenten die gebruik maken van deze component of verwijzen aan deze component aangepast (bij een domeincomponent) of verwijderd worden (bij andere componenten).

Je moet dus een duidelijk inzicht hebben op de impact van 'Cascade' en weten waarmee je bezig bent als je deze optie gebruikt, anders riskeer je onaangename neveneffecten.

Waartoe dienen acties bij vreemde sleutels? Wat is het verschil tussen NO ACTON en RESTRICT? – examen— On delete en on update durft hij wel vragen op het examen zegt hij.

Deze acties bepalen wat er moet gebeuren met de waarde van een vreemde sleutel (waarvoor de actie gedefinieerd is) wanneer de primaire sleutel waaraan deze vreemde sleutel verwijst, aangepast of verwijderd wordt. (Acties zoals ON DELETE en ON UPDATE (beide optioneel)).

No action: maakt dat er een fout gegenereerd zal worden door het dbms van zodra er een vraag is om de primaire sleutelwaarde waaraan verwezen wordt aan te passen of te verwijderen. De aanpassing of verwijdering zal NIET gebeuren. Dergelijke foutcodes kunnen worden opgevangen in applicatieprogramma's (m.a.w. je kan als programmeur zien dat er zich een probleem voordoet en dit ook gepast opvangen in de applicatiecode).

Restrict: maakt dat een vraag om de primaire sleutelwaarde waaraan verwezen wordt aan te passen of te verwijderen, verhinderd zal worden door het dbms. Er wordt daarbij geen foutcode gegenereerd. De afblokking blijft transparant voor de toepassingsprogramma's, waardoor een programmeur dit ook niet kan opvangen in de applicatiecode.

Verwerking van DML

Waarom is het belangrijk dat elke SQL-query wordt omgezet naar een algebraïsche expressie? Wat kunnen jullie daar zelf uit leren?
Info: Er wordt niet zo vaak iets over gesteld op het examen, maar adhv vertalingstabellen moet je de bijhorende algebraïsche expressie kunnen schrijven, oefening hierover kan op examen (zie verwerking DML in slides)

Door de omzetting naar een algebraïsche expressie weet het dbms exact welke operatoren het moet uitvoeren, in welke volgorde op welke (tijdelijke) basisrelaties.

Het uitrekenen (evalueren) van algebraïsche expressies is volledig geïmplementeerd in het dbms. Daardoor kan je als gebruiker rechtstreeks SQL queries doorgeven aan het dbms zonder dat je hierbij enige code hoeft te schrijven in een programmeertaal. Alle logica die nodig is om de query op te lossen zit dus vervat in de SQL-query.

Je kan hiervan gebruik maken om zelf beter SQL te begrijpen: als het niet duidelijk is waarom een SQL-query een bepaald resultaat oplevert kan je de query zelf omzetten naar een algebraïsche expressie.

Waarom kan een DBMS maar een beperkte tijd spenderen aan queryoptimalisatie? Wat kan je hier als DBA tegen doen?
Het optimaliseren en uitvoeren van de geoptimaliseerde query mag niet meer tijd in beslag nemen dat het uitvoeren van de niet-geoptimaliseerde query. Uitzondering: queries die meerdere keren uitgevoerd worden. (cruciale queries)

Voor kritische/belangrijke queries kan de dba kijken naar het queryplan en dit desgevallend manueel aanpassen of laten aanpassen door het dbms door bv. mee te geven dat bepaalde indexen moeten gebruikt worden of er meer tijd aan optimalisatie kan worden besteed. Het werken met queryplannen valt buiten de SQL standaard en is daarom sterk systeemgebonden. De betere dbms'sen zullen een betere ondersteuning bieden.

Query-by-example

Wat zijn de voor- en nadelen van het werken met een QBE-techniek?
= Visuele representatie van databankschema, adhv een interface mogelijkheid om attributen te selecteren.

Voordelen: je hoeft geen SQL-taal te kennen om te werken met een databank. Je geeft aan hoe het resultaat van de query er moet uitzien -- niet hoe het resultaat moet worden berekend.

Nadelen: voor complexere queries is de QBE-techniek niet handig. De QBE-techniek kan niet worden gebruikt om een databank aan te spreken vanuit een toepassingsprogramma.

Hoofdstuk 7: Objecttechnologie (week 8)

Enkel inleiding, SQL2011 (ODMG dus niet!)

Objectdatabanktechnologie

Waarom is de 'impedance mismatch' nog niet opgelost door objectdatabanktechnologie?
Men dient nog steeds een omzetting te maken tussen de datastructuren/datatypes van de OO-programmeertaal en de datastructuren/datatypes van het databankmodel.

Bij ODMG moet er een omzetting gebeuren naar ODL.
Bij SQL:2011 moet er een omzetting gebeuren naar basisrelaties die nu wel attributen met complexere structuur en methoden kunnen bevatten.

Waarom mag objectpersistentie niet bepaald worden door het objecttype?
Voor een gegeven objecttype moeten er zowel persistente als transiënte objecten kunnen bestaan, onafhankelijk van de klasse. Indien objectpersistentie zou worden bepaald door het objecttype zouden de objecten van dat type ofwel allemaal persistent, ofwel allemaal transiënt zijn.

Waarom volstaat het naamgevingsmechanisme niet om persistentie af te dwingen?
-- vraag van herexamen 2020 -- (min 9)
Men zou dan voor elk object een unieke naam moeten bedenken. Voor heel grote databanken is dat heel omslachtige taak. (Zeker als die naam ook betekenisvol moet zijn.)

Naamgevingsmechanisme: als een object een unieke naam heeft, wordt het object persistent gemaakt (conventie). In de praktijk ook echter nood aan bereikbaarheidsmechanisme: anders zou te veel dingen manueel uniek moeten benoemen.

Wat is het voordeel om te werken met objectidentifiers?
De waarde van een objectidentifier kan niet veranderen tijdens de levensduur van het object. Het object kan dus m.a.w. niet van identiteit veranderen. Het zal in de toekomst ook niet hergebruikt worden. Als het dbms met een gegeven objectidentifier naar een object verwijst, is het zeker dat het steeds naar hetzelfde object verwijst.

Je kan sneller zoeken door de tuple-identifiers (geen joins meer nodig) bij relationele gegevensbanken.

SQL:2011

Waarom zijn er geen typeconstructuren voorzien voor sets?
Sets zijn speciale gevallen van multisets. Elke set is een multiset waarbij elk element juist 1 keer voorkomt. SQL:2011 beschikt over een 'IS A SET' operator die kan gebruikt worden voor het opbouwen van selectiecondities in queries. Deze operator levert 'waar' op als de multiset een set is.

Wat is de meerwaarde van een referentieattribuut?
Referentieattributen laten het toe om vanuit een basisrelatie rechtstreeks te refereren aan de tuple-identifier van een tuple van een (andere) basisrelatie.
Hierdoor zijn er geen join-operaties meer nodig.

Welke vormen van overerving worden ondersteund? Welke mogelijkheden zie je voor het dbms bij overerving bij de aanpak van relaties?

1. Overerving op het niveau van het relatieschema.

Men kan bij de definitie van een gestructureerd type opgeven dat dat type één of meerdere oudertypes heeft (met UNDER-clause). Als dit zo is erft het type alle karakteristieken van deze oudertypes (meervoudige overerving). Door basisrelaties op te bouwen met deze gestructureerde types kan men dus overerving tussen extensies van basisrelaties vastleggen.

2. Overerving op het niveau van de extensie van een basisrelatie.

Dit kan je zien als een horizontale fragmentatie en replicatie van de extensie van de ouderbasisrelatie (enkelvoudige overerving). Elk tuple van een kindbasisrelatie komt noodzakelijk ook voor als tuple van de ouderbasisrelatie. Het dbms moet ervoor zorgen dat het mechanismen heeft om deze overlappende extensies consistent te houden.

Logisch databankontwerp

Waarom is een zwakkere definitie van de eerste normaalvorm noodzakelijk?

Zonder deze zwakkere definitie wordt elk voordeel van het gebruik van een niet-atomair datatype (dat toegestaan is door SQL:2011) meteen tenietgedaan bij het afdwingen van de eerste normaalvorm bij normaliseren en dat is uiteraard niet de bedoeling.

Wat is een ORM-systeem? Welke voor- en nadelen zijn er bij het werken met een ORM-systeem?

Een ORM-systeem is een tool voor het automatisch omzetten van objecten uit een programma naar tuples van basisrelaties van een relationele databank die door de tool is gegenereerd. Daarnaast voorziet de tool ook in een API waarmee je vanuit je programma de relationele databank kan bevragen.

Bv.

```
Artiest a = Artiest.Get(Artiest.Properties.A_ID == 'A01');
```

i.p.v.

```
SELECT * FROM Artiest WHERE A_ID='A01';
```

Voordeel: je hoeft als programmeur geen databank te ontwerpen en geen SQL queries te schrijven.
Nadelen: ORM werkt alleen goed voor data die met een eenvoudig databankschema kan worden gemodelleerd. Geavanceerde queries worden meestal niet ondersteund.

Hoofdstuk 9: Beveiliging (Week 10)

(Week 9 niets: hoofdstuk 8 is niet te kennen voor Relationele Gegevensbanken, wel voor Databanken)

Code-injectie

Wat kan je doen tegen code-injectie?
Logboek kan u daar niet tegen beschermen, maar kan je eigen gebruikers wel ontraden om dat te doen. Maar code-injectie gebeurt meestal op afstand.

Als applicatieontwikkelaar: implementeer afdoende controle op de invoer van gebruikers alvorens deze invoer te gebruiken bij de preparatie van SQL-instructies.

Als DBA: Werk met afdoende privilege-lijsten die dan gekoppeld zijn aan gebruikersaccounts of gebruikersprofielen. Merk wel op dat privileges nog vrij algemeen zijn en dus beperkingen hebben. Zo heb je bv. geen privilege dat het toelaat om slechts een beperkt aantal tuples van een basisrelatie te zien. Enige creativiteit met views kan daarbij helpen, maar biedt ook geen sluitende oplossing.

Hulpmiddelen tegen ongeoorloofd gebruik

Waarom is 'flow'-controle belangrijk?
'Flow'-controle laat toe om inzicht te krijgen in welke sequenties van instructies één of meerdere gebruikers doorgeven aan het dbms. Wanneer ze dat doen en met welke frequenties ze dat doen. Hieruit kunnen verdachte patronen worden afgeleid. (Manueel door een ervaren DBA of met behulp van technieken zoals bv. machine learning).

Hacking

Wat zou je zelf doen als je hacking detecteerde en waarom?
Er zijn verschillende soorten strategieën mogelijk. Er moet wel worden voorkomen dat een databank onherstelbaar beschadigd raakt. De identificatie van de hacker kan een belangrijk (maar niet noodzakelijk) aspect zijn. Om die reden kan het soms nuttig zijn om onder gecontroleerde omstandigheden de detectie van de hack nog niet direct kenbaar te maken, zodat meer informatie kan worden verzameld. Bedrijven hebben er ook niet altijd baat bij om een geslaagde hacking naar buiten toe te communiceren.

Authenticatie

Welk probleem zie je bij biometrische authenticatie? Wat kan men hiertegen doen?
Als de identiteit gestolen wordt kan je geen nieuwe meer instellen.

Biometrische data zijn uniek. Het is quasi onmogelijk om iemand andere biometrische kenmerken te geven. Biometrische authenticatie gaat daarom best uit van gemodificeerde gedigitaliseerde biometrische data zodat bij diefstal van de gedigitaliseerde data een andere modificatietechniek het probleem kan verhelpen.
Biometrische data zoals vingerafdrukken zijn ook gevoelig aan veranderingen ten gevolge van omstandigheden (bv verwonding).

Toegangsbeperking

Waarom werkt men met privileges en niet met restricties?
Een privilege vergeten toekennen heeft minder schadelijke gevolgen dan een restrictie vergeten toekennen. In het eerste geval kan de gebruiker zijn/haar taken niet uitvoeren en zal hierover direct gecommuniceerd worden. In het tweede geval kan de gebruiker taken uitvoeren die hij/zij normaal gezien niet mag uitvoeren en kan dat stilgehouden worden.

Wat is het probleem met verlaten privileges? Wat kan men hiertegen doen?
Een verlaten privilege zou verkregen zijn door een gebruiker die zelf het privilege niet meer bezit. Dit wordt gezien als een ongewenste situatie. De persoon die het privilege doorgaf werkt misschien niet langer meer bij het bedrijf, of heeft misschien het vertrouwen van het bedrijf geschonden.

Men moet hiertegen niets doen. Het dbms voorkomt dat verlaten privileges kunnen ontstaan (cf. Restrict met foutmelding of Cascade).

Wat is het probleem bij overlappende privileges? Wat kan men hiertegen doen?
Overlappende privileges zijn gevaarlijk omdat het dbms hier geen controle op uitvoert. Krijgt met bijvoorbeeld toegang tot om een volledige schilderkunstdatabank te lezen via een gebruikersprofiel 'Gast' en krijgt men daarnaast expliciet een privilege om de basisrelatie 'eigenaar' uit deze databank te lezen. Dan zal het intrekken van deze laatste privilege geen enkele impact hebben en kan de gebruiker de basisrelatie 'eigenaar' nog lezen via het 'Gast'-gebruikersprofiel.
Het enige wat men hiertegen kan doen is privileges nauwgezet toekennen zodat overlap zo veel als mogelijk wordt vermeden.

Wat wil de stereigenschap zeggen? Waarom is deze belangrijk?
De stereigenschap zegt dat je bij het werken met beveiligingsniveaus en toegangsniveaus enkel data kan schrijven waarvan het beveiligingsniveau groter (strikter) dan of gelijk is aan jou toegangsniveau.

Dit laat het toe dat je zaken schrijft die je nadien in principe niet meer kan lezen.

Databankbeheersystemen zullen de werking van 'meer-niveau'-relaties echter transparant houden voor de gebruikers. M.a.w. een gebruiker ziet niet op welk beveiligingsniveau de data op zijn/haar toegangsniveau of hoger staat.

Dat wil zeggen dat je in het voorbeeld op toegangsniveau V eigenaar Boijmans kan aanpassen, terwijl Eigenaar Louvre niet kan aangepast worden en een foutmelding zal opleveren. De waarde van 'Mona Lisa' overschrijven op beveiligingsniveau G zal in de praktijk nooit echt lukken. Omwille van de transparantie blijft deze waarde onzichtbaar en aanpassing van de waarde 30.000.000 op beveiligingsniveau V zal resulteren in het overschrijven van deze waarde, wat zichtbaar zal zijn voor iedereen met toegangsniveau V of hoger.

Waarom heeft men bij het werken met beveiligingsniveaus nood aan filtering resp. poly-instantiatie?
Filtering en poly-instantiatie zijn nodig om het werken met 'meer-niveau'-relaties transparant te houden voor gebruikers.

Zonder filtering zou het direct zichtbaar zijn dat er andere data beschikbaar zijn op een hoger niveau.

Zonder poly-instantiatie kan men geen data op een transparante manier aanpassen zonder data op een hoger beveiligingsniveau mee aan te passen. Afblokken van een dergelijke aanpassing zou de gebruiker een duidelijk signaal geven dat er data op een hoger beveiligingsniveau aanwezig zijn.

Waarom is statische bevraging op zich geen afdoende beveiligingsstrategie? Wat zijn de nadelen van query restriction, resp. data-swapping?
Gevaar voor covert channels.

Men kan door het louter toepassen van voldoende statistische queries altijd een (individuele) tracker bouwen (oorspronkelijke waardes afleiden). Technieken die voorkomen dat de benodigde statistische queries kunnen worden uitgevoerd maken de databank onbruikbaar.

Query restriction beperkt het aantal queries dat een gebruiker mag uitvoeren binnen een bepaalde periode. Als die gebruiker voldoende geduld heeft kan hij/zij die queries toch uitvoeren.

Data-swapping heeft tot doel om bij individuele tracking verkeerde data terug te geven. Deze data worden dan bewust verkeerde opgeslagen in de databank, waardoor de databank voor een groot stuk onbruikbaar wordt (niemand kan de juiste data nog achterhalen).

Auditbestanden

Wat zijn 'before' – en 'after'-images? Waarvoor worden deze gebruikt?
Grote uitdaging voor constructeur van databankbeheersysteem. Zo compact mogelijk.

'Before'- en 'after'-images zijn beknopte beschrijvingen van het deel van de databank dat bij een SQL-instructie wordt aangepast. Ze laten respectievelijk toe om snel te reconstrueren hoe dat deel er uitzag voor de uitvoering en na de uitvoering van de instructie.

Een 'before'-image kan worden gebruikt om een instructie ongedaan te maken en de databank te herstellen naar haar toestand voor de uitvoering van de instructie.

Een 'after'-image kan worden gebruikt om de databank snel te herstellen naar haar toestand na de uitvoering van een instructie zonder dat daarbij de instructie opnieuw moet worden uitgevoerd. Dit komt nog verder aan bod in het hoofdstuk rond herstel na falen.

Versleutelen

Waarom versleutel je best enkel de meest gevoelige data?
Versleutelen is een heel tijdrovende en dus vertragende taak.

De data moeten worden versleuteld voordat ze in een diskblock op een opslagmedium worden weggeschreven en opgeslagen data moeten worden gedecodeerd vooraleer ze worden ingeladen in de databankbuffers om daarna verder de worden verwerkt bij de queryverwerking.

Hoe minder data je gaat versleutelen, hoe sneller je kan werken.

Wat zijn de voor- en nadelen van het werken met symmetrische sleutels?
Symmetrische sleutel zijn minder veilig maar zorgen voor een snellere codering en decodering. Als een dbms zelf data kan versleutelen dan is dat via een symmetrisch algoritme.

Versleuteling met een asymmetrisch algoritme kan enkel via derde partij software.

Men kan ook versleutelen op het niveau van het besturingssysteem. In dat geval wordt een volledig recordbestand versleuteld en kan je de versleuteling niet beperken tot gevoelige attributen. (Een databank wordt weggeschreven in één of meerdere recordbestanden.)

Hoofdstuk 10: Herstel na falen (week 11)

Transacties

Hoe kan de duurzaamheidseigenschap van transacties worden gegarandeerd?
Als databaseadministrator moet je back-up maken en logbestanden laten maken.

De DBA is verantwoordelijk voor het nemen van back-up- en logbestanden en dient dat te doen zodat er geen data verloren kan gaan, ook in uitzonderlijke omstandigheden zoals bv. brand.

Het DBMS ondersteunt de herstelprocedures bij 'soft crashes' en 'hard crashes'.

Waarom zijn synchronisatiepunten nodig?
Synchronisatiepunten zijn nodig bij het gebruik van expliciete transacties. Het DBMS dient bij een rollback van een impliciete transactie te weten naar welke consistente toestand moet worden teruggekeerd. Bij een impliciete transactie die binnen een expliciete transactie is opgenomen moet men terugkeren naar de recentste consistente toestand voor de start van de expliciete transactie.

Synchronisatiepunten worden geplaatst bij de start van de eerste expliciete transactie en daarna bij elke commit van een expliciete transactie en bij elke commit van een impliciete transactie die niet is opgenomen in een expliciete transactie.

Waarom gebruik je best geen savepoints? Welke eigenschap van transacties komt dan gedrang?
Savepoints zorgen ervoor dat niet langer voldaan is aan de atomiciteitseigenschap van de expliciete transactie waarin de savepoint is opgenomen. Hierdoor kunnen er zich situaties voordoen waarbij het consistent gebruik van de data niet is gegarandeerd. (Na rollback tot aan een savepoint dient de expliciete transactie hernomen te worden vanaf de savepoint, maar heb je geen garantie dat deze uitvoering correct kan verlopen. Uiteindelijk kan je dan een situatie hebben waarbij slechts een deel van de expliciete transactie is gecommit wat consistentieproblemen kan opleveren.)

Falen

Wat is het verschil tussen een 'soft crash' en een 'hard crash'? Waarom wordt er zo veel aandacht besteed aan het herstel na 'soft crash'? Bij een 'soft crash' gaat enkel de inhoud van de databankbuffers verloren. Omdat data in deze buffers kunnen aangepast zijn door gecommitte transacties, kan hierdoor een probleem met de duurzaamheid van deze transacties. Het DBMS lost deze problemen op met een herstelprocedure voor 'soft crashes'.

Mochten we hier geen aandacht aan besteden dan zouden we een groot deel van de aanpassingen sinds de recentste rollback verliezen en daarbij niet meer weten wat wel en wat niet verloren gegaan is. Dit zou de databank inconsistent maken, wat moet worden voorkomen.

Waarom volstaat een RAID-systeem niet als bescherming tegen falen? Bij een RAID-systeem kunnen er meer schijven tegelijkertijd kapotgaan dan de RAID-configuratie kan opvangen. (Bij de meeste configuraties is men slechts beschermd tegen het uitvallen van 1 schijf.) Als meer schijven uitvallen dan het RAID-systeem kan opvangen, dan heeft men informatieverlies en moet er een herstelprocedure voor 'hard crashes' worden uitgevoerd. Daartoe heb je een back-up- en logbestand nodig.

Mogen logbestanden en back-upbestanden op hetzelfde opslagmedium worden opgeslagen?
Logbestanden en back-upbestanden mogen op hetzelfde opslagmedium worden geplaatst. Als dit medium uitvalt, heeft men nog steeds een werkende databank en kan er een nieuwe back-up worden genomen. Na elke back-up wordt gestart met een nieuw, leeg logbestand.

Databanken en logbestanden mogen niet op hetzelfde opslagmedium worden bewaard. Als dit opslagmedium faalt, ben je niet alleen je databank, maar ook je logbestand kwijt en kan je niet meer herstellen.

Herstel

Waarom is 'steal, no force' de meest gebruikte 'flushing'-strategie?
Steal = op het moment dat je gaat flushen, mogen er nog transacties bezig zijn.

Steal, no force is de efficiëntste strategie: het DBMS hoeft niet te wachten en/of transacties uit te stellen zodat er een moment ontstaat waarop geen transacties bezig zijn en het DBMS moet niet bij elke commit een vertragende flushing uitvoeren. (Na flushing zijn de databankbuffers leeg, waardoor transacties die bezig zijn, moeten wachten totdat hun data opnieuw is ingeladen om verder te kunnen.)

Welke eigenschap van transacties is zonder de 'write-ahead log'-regel niet gegarandeerd?
Zonder 'write-ahead log'-regel is de duurzaamheidseigenschap van transacties niet gegarandeerd: als de transactie gecommit is, maar de commit op het moment van falen niet is weggeschreven naar het logbestand, bestaat er geen enkele manier om nog te achterhalen dat deze transactie gecommit was en gaat deze alsnog verloren.

Waarom werden controlepunten ingevoerd? Controlepunten laten het toe om herstel na 'soft crash' te versnellen. Bij een controlepunt is er een synchronisatie tussen de databankbuffers en de databank in secundair geheugen, waardoor we zeker zijn dat deze databank consistent is en we bij een herstel kunnen vertrekken vanaf het recentste controlepunt. Zonder controlepunten moet het volledig logbestand bij een herstel verwerkt worden wat bij databanken met heel veel activiteit (transacties) en dus veel registraties in het logbestand heel tijdrovend kan zijn.

Hersteltechnieken met uitgestelde aanpassing

Wat is de rol van de databankbuffers bij hersteltechnieken met uitgestelde aanpassing?
Databankbuffers zijn altijd nodig opdat een processordata snel zou kunnen lezen.

Het is niet omdat er niet naar de databankbuffers wordt geschreven dat deze overbodig worden.

Waarom is er nog flushing nodig bij deze techniek? Bij het lezen worden de databankbuffers gevuld en raken deze ook vol. Op een bepaald moment zal er een flushing nodig zijn om nieuwe diskblocks te kunnen inlezen.

Waarom moeten de 'after images' weggeschreven worden bij een commit van een transactie (en niet later)? After images worden best weggeschreven bij de commit als onderdeel van het klaarmaken van een transactie om gecommit te worden. Na het wegschrijven is de transactie dan klaar om gecommit te worden en kan de commit met de 'write-ahead log'-regel worden weggeschreven.

Wachten we langer en worden de after images weggeschreven nadat de commit is afgerond, dan riskeren we dat een andere transactie de gecommitte data wil lezen en nog steeds de oude waarden te zien krijgt.

Waarom moeten bij herstel de 'after images' (opnieuw) naar de databank worden geschreven? Dit is om zeker te zijn dat de databank na het herstel opnieuw consistent is.

Het kan immers gebeuren dat de after images zijn weggeschreven maar de commit van de transactie niet is weggeschreven in het logbestand. Door te vertrekken van de consistente toestand bij het controlepunt en enkel de after images van de gecommitte transacties terug te schrijven, wordt dit opgelost.

Databank en logbestand mogen niet op zelfde schijf staan, dan kan je enkel herstellen vanaf back-up of zo…

Hersteltechnieken met onmiddellijke aanpassing

Waarom worden de 'undo'- en 'redo'-lijsten pas aangemaakt bij het herstel? Het continu bijhouden van een 'undo'- en 'redo'-lijst zou de herstelprocedure kunnen versnellen, maar zou tegelijk de uitvoering van de transacties vertragen daar het DBMS extra tijd zou moeten besteden aan het plaatsen en verplaatsen van transacties in deze lijsten. Het is pas wanneer er zich een soft crash voordoet dat het DBMS-resources gaat besteden aan het herstel. Het hoofddoel blijft immers: transacties zo efficiënt mogelijk uitvoeren om zo gebruikers en applicatieprogramma's zo snel mogelijk te bedienen.

Schaduwpagina’s

Wat is de rol van de databankbuffers bij het werken met schaduwpagina's? Databankbuffers zijn altijd nodig opdat een processordata snel zou kunnen lezen. Het is niet omdat er niet naar de databankbuffers wordt geschreven dat deze overbodig worden.

Vroegere examenvraag: Hoeveel keer moet je door logbestand gaan?
Antwoord (boek blz 319): 2 keer:
- eerst van achter naar voren, stop bij recentste controlepunt, registreet welke transacties bevestiging (commit) en startinstructie zijn gevonden.
- Doorloop nog eens, van recentste controle punt tot het einde. Voor elke geregistreerde transactie worden de wijzigingen (opnieuw) uitgevoerd.

Hoofdstuk 11: delen van gegevens (week 12)

Wat is het verschil tussen 'interleaved' en parallelle verwerking van transacties?
Bij 'interleaved' verwerking heb je te maken met een processor die meerdere transacties bedient door afwisselend een stukje processortijd te besteden aan elk van deze transacties. Zo 'zien' alle transacties voortgang en lijkt het alsof alle transacties tegelijkertijd worden verwerkt.

Bij parallelle verwerking heb je meerdere processoren die de voorgaande werkwijze volgen. Deze processoren werken tegelijkertijd zodat de transacties ook effectief tegelijkertijd worden verwerkt.

In de cursus worden alle technieken geïllustreerd via "interleaved". In de praktijk gaat men doorgaans meer transacties tegelijkertijd uitvoeren dan er processoren aanwezig zijn.

Welk voordeel heb je als transacties serialiseerbaar zijn? Hoe gaan de 'timestamping'- en 'locking'-methoden om met dit concept?
Van serialiseerbare transacties weet je met zekerheid dat ze zonder concurrency problemen worden uitgevoerd.

Niet elke interleaved verwerking van transacties is serialiseerbaar. 'Timestamping' en 'locking' zorgen ervoor dat er ingegrepen wordt op de volgorde waarin instructies uit transacties worden uitgevoerd door resp. transacties te rollbacken of te laten wachten en voorrang te geven aan andere transacties. Door deze ingrepen wordt gegarandeerd dat de verwerking van de transacties (uitvoeringsvolgorde van de instructies) serialiseerbaar is, wat op zijn beurt impliceert dat er zich geen concurrency problemen kunnen voordoen.

Waarom moet je opletten als je werkt met isolatieniveaus die lager zijn dan 'serializable'? Waarom zijn deze lagere niveaus in de praktijk toegestaan?
Als je werkt met een lager isolatieniveau heb je een risico op concurrency problemen. Concurrency problemen moeten voorkomen worden

Een DBA mag enkel gebruik maken van een lager isolatieniveau als de gebruikscontext van het DBMS zo is dat er zich geen concurrency problemen kunnen voordoen (bv. 1 gebruiker of enkel gebruikers die kunnen lezen).

Door te werken op een lager isolatieniveau zal het DBMS sneller werken. Het is uitermate belangrijk om attent te zijn voor een veranderende gebruikscontext.

Waarom dien je bij het basisprotocol voor het lezen de maximum-operator te gebruiken voor het actualiseren van de 'lees-timestamp'?
Het protocol voor lezen sluit niet uit dat een jongere transactie het gegeven gelezen heeft. De lees-timestamp moet de timestamp van de jongste transactie die het gegeven las bevatten. Dus als een jongere transactie het gegeven las, moet de recentere lees-timestamp van deze transactie behouden blijven, wat gebeurt door de maximum-operator te gebruiken.

Welk voordeel levert het toepassen van de aanpassingsregel van Thomas op?
De aanpassingsregel van Thomas zorgt ervoor dat de transactie niet moet worden gerollbackt. De aanpassing wordt niet uitgevoerd binnen de transactie en de transactie kan committen. Nadien wordt een nieuwe transactie opgestart om de aanpassing en de rest van de originele transactie af te werken. Door het voorkomen van de rollback gaat er geen werk verloren.

Wat is het probleem bij livelocks? Hoe kan je een livelock detecteren?
Bij een livelock blijft een transactie wachten op een lock omdat andere transacties steeds de lock krijgen van het dbms wanneer de lock vrijkomt.

Een livelock kan worden gedetecteerd met een telling van het aantal keer dat een andere transactie de voorkeur kreeg. Naarmate de teller groter wordt kan de prioriteit van de transactie verhoogd worden, zodat deze uiteindelijk wel uitgevoerd zal worden.

Waarom gebruikt men bij deadlockpreventietechnieken een techniek van veroudering?
Door de veroudering wordt de kans kleiner dat dezelfde transactie opnieuw zal moeten worden gerollbackt bij een volgende controle.

Telkens de jongste transactie die wordt afgebroken: afbreken van oudere transactie laat meer CPU-tijd verloren gaan.

Oudere wacht nooit op een jongere. De jongere opstarten met oudere timestamp waardoor hij ouder wordt.

Waarom werken optimistische methoden enkel efficiënt als vrijwel alle transacties slagen voor de validatietest?
Het rollbacken van een volledige transactie gaat doorgaans gepaard met het verlies een grote hoeveelheid werk (dat daarna opnieuw moet worden uitgevoerd). Dit betekent tijdsverlies.

Een timestamping-methode gaat ook transacties rollbacken, maar dan meestal in een veel vroegere fase waardoor het tijdsverlies veel beperkter is.

Het tijdsverlies door het wachten bij een locking-methode is doorgaans ook veel beperkter.

Optimistische methoden werken enkel efficiënt wanneer er maar heel weinig transacties (volledig) moeten worden gerollbackt.

Hoofdstuk 12: NoSQL (week 12)

Big data

Wat is big data?
Big data slaat op elke situatie waarbij het databeheer niet met een conventioneel DBMS kan gebeuren. Veel voorkomende problemen hebben te maken met:

- Volume (=big data)
- Variëteit (=varied data)
- Snelheid (=fast data)
- Waarheidsgetrouwheid/datakwaliteit (= bad data)

Als je met minstens één van deze problemen wordt geconfronteerd heb je te maken met 'big' data.

Waarom vormt het variëteitsprobleem een grotere uitdaging dat het volumeprobleem?
Het volumeprobleem valt al vrij goed op te lossen via horizontale schaling. Conventionele DBMS's beginnen ook al horizontale schaling te ondersteunen, maar dan met focus op dataconsistentie. Hierdoor kan de beschikbaarheid van de data in het gedrang komen. (Als er replica's zijn en er moet gewacht worden tot al deze replica's na een aanpassing zijn gesynchroniseerd om verder te kunnen werken).

Het variëteitsprobleem vormt een grotere uitdaging omdat je dan dient over te stappen naar een schemaloze databank. Dat heeft vaak een grotere impact en wordt doorgaans als moeilijker ervaren. (Let wel het is aangewezen om te streven naar een polyglot oplossing -- zie verder.)

Wat kan je doen tegen het snelheidsprobleem? Welke problemen krijg je dan meestal in de plaats?
Een snelheidsprobleem kan meestal alleen worden opgelost door een lichtgewicht NoSQL dbms te gebruiken. Een dergelijk systeem hoeft geen tijd te spenderen aan tijdrovende dataconversies, integriteitscontroles, concurrencycontrole mechanismen, etc.

De prijs die je doorgaans betaalt is dat de consistentie van je databank niet meer gegarandeerd wordt door het dbms en je zelf verantwoordelijk wordt voor een aantal taken die anders door het dbms worden uitgevoerd.

NoSQL-databanken

Wat wordt bedoeld met een polyglot databanksysteem? Met een polyglot databanksysteem bedoelt men dat er meerdere DBMS'sen gebruikt worden om data te beheren. De datacollectie wordt dan opgesplitst over de verschillende DBMS'sen zodat DBMS optimaal kan worden ingezet om die taken te doen waar het goed in is.
Zo splits je bijvoorbeeld doorgaans je 'big' data af in NoSQL systeem, maar behoud je je data die geen problemen opleveren in een SQL (relationeel) systeem.

Wat is het belang van het CAP-theorema? Kan je dit probleem voorkomen?
Het CAP-theorema stelt dat als je gebruik maakt van horizontale schaling, je ofwel consistentie, ofwel beschikbaarheid van data moet opgeven.

Dit probleem kan niet worden voorkomen en is eigen aan het feit dat bij horizontale schaling een tijdrovende synchronisatie nodig is om replica's consistent te houden. Als je de data vrijgeeft tijdens de synchronisatie is deze beschikbaar maar niet noodzakelijk consistent. Als je dit niet doet is ze niet beschikbaar.

Waarvoor staat BASE? Wat wordt bedoeld met 'Eventual consistent'?
BASE karakteriseert NoSQL data. Eventual consistent slaat op het feit dat de data beschikbaar is (basically available), maar niet noodzakelijk consistent. Je hebt wel de garantie dat het dbms er alles aan doet om de consistente na synchronisatie in de toekomst te herstellen. Dat laatste is wat wordt bedoeld met 'eventual consistent'.

Wat is volgens u het grootste nadeel wanneer je gebruik maakt van een NoSQL databanksysteem?
Het grootste nadeel is dat er meer verantwoordelijkheid bij de toepassingsontwikkelaars komt te liggen. Complexe queries kunnen niet worden afgehandeld door het dbms, integriteitscontrole evenmin, ...

Wanneer zou je gebruik maken van een key-value store, wanneer van een document store'?
Een key-value store gebruik je doorgaans wanneer je data heel snel moet kunnen wegschrijven (en er geen tijd is voor datatransformaties). In een document store worden de data beperkt geïnterpreteerd door het DBMS dat om kan gaan met JSON- of XML-formaten. Hierdoor weet het DBMS welke de attribuutnamen zijn en wat de bijhorende attribuutwaarden zijn, wat kan worden aangewend bij een bevraging. Een document store levert dus betere bevragingsfaciliteiten op.

Waarom zijn column stores minder flexibel m.b.t. het behandelen van het variëteitsprobleem dan document stores? Bij een column store worden attributen van verschillende entiteiten samen weggeschreven in eenzelfde kolomfamilie. Bij een document store heeft men een apart document voor elke entiteit. Dit maakt het opvragen en behandelen van entiteiten in een document store eenvoudiger. Hierdoor zijn document stores, met betrekking tot het behandelen van het variëteitsprobleem, flexibeler dan column stores.

Welk probleem heb je als je een graafdatabank opslaat in een key-value store?
Het belangrijkste probleem met graafdatabanksystemen is een efficiënte horizontale schaling. Marktleider Neo4J maakt bv. gebruik van sharding, maar deze werkt minder efficiënt (Neo4J werkt het beste op een systeem met verticale schaling.)

Andere systemen trachten het volumeprobleem op te lossen door een key-value store te gebruiken voor de opslag van de grafen. Dit werkt uiteraard ook vertragend, want introduceert een extra modelleringslaag.

Graafdatabanksystemen vormen dus geen goede oplossing voor een volumeprobleem. Je gebruikt ze vooral wanneer je sterkt gelinkte data hebt (veel onderlinge links tussen entiteiten) en wil werken zonder vast databankschema (variëteitsprobleem).