Samenvatting Databanken

0.1 Voorwoord

Dit hoort bij “Principes van Databases” door Guy De Tré die het vak “Databanken” doceert aan de UGent. Academiejaar 2010-2011, zittijd januari. De meeste volzinnen uit deze samenvatting komen dan ook rechtstreeks uit dit boek of zijn een parafrasering ervan. Ik wens dan ook niet te beweren de volledige auteursrechten van dit document te bezitten.
Samenvatting nogal fragmentarisch, heb zelf al boek gelezen dus heb dan ook vaak geen volzinnen nodig. Indien begrippen te fragmentarisch of beknopt, leest zelf et boek é, ge meugt ol blie zien daj da ier kriegt va min. Aja, en sorry voor het west-vlaams her en der. Welcome to Ghent, get used to it. Grtz, Reuse.

1. Hoofdstuk 1

Data: eentjes en nulletjes (of strings, integers…) zonder context
Informatie: iets wat perfect te reconstrueren is naar volzinnen. (<Voornaam> <Naam> is geboren op <Geboortedatum> in <Geboortestad> …)

Een **database** is een collectie van persistente data.
En computersysteem voor het beheer van een database wordt een **databasesysteem** (DBS) genoemd.
Een databasesysteem bestaat uit hardware, data en software. De software van een DBS wordt een **Databasemanagementsystem** (DBMS) genoemd.
Hardware: primair geheugen/secundair geheugen… voor een informaticus triviaal
Een recordtype is een collectie van records met dezelfde betekenis van de velden (“tabel” in een database).

Een **datawarehouse** is een grote database die bedoeld is voor data-analyse. Meestal bevat het de geschiedenis van één database. Vaak wordt hier regressie op uitgevoerd maar ook andere statistische analyses zijn mogelijk, en dit heet **datamining**.

Gebruikersprofielen: **Data-Administrator** (DA): centrale verantwoordelijkheid voor de data, hoge functie in bedrijf. De DA beslist ondermeer welke data, in welk formaat, in welke database moet worden opgeslagen (en ontwerpt dus ook de database). Cruciaal voor grote conversies of aanpassingen. Wordt vaak door één iemand of een zeer klein team uitgevoerd.

Gebruikersprofielen: **Database-Administrator** (DBA): technisch verantwoordelijk voor het (goed, snel…) werken van de DB, backups nemen (en evt DB restoren adhv bu), implementatie van de beslissingen van de DA’s… Lage functie in bedrijf, meerstal verspreid over verschillende personen.

Gebruikersprofielen: **Toepassingsontwikkelaar**: staat in voor de ontwikkeling van interactieve, gebruiksvriendelijke manieren om noobs met de DB te laten werken. Betreft vaak views en/of elementaire sites. Meestal een functie binnen het bedrijf zelf.

Gebruikersprofielen: **Eindgebruiker**: noobs die ni ganse dagen zitten te moosen met de DB maar af en toe kji checkn. Buiten de "**geavanceerde eindgebruikers**", die krijgen rechtstreeks toegang tot de database omda ze der twuk van kennen en omda ze hele dagen (blijkbaar) ni anders te doen ebn dan rechtstreeks de DB te manipuleern.

Databasemanagementsystem: Hoofdfunctionaliteiten:
Databasedefinitie.
Databasemanipulatie.
Databasereconstructie.
(allemaal triviaal indien boek gelezen)

Databasemanagementsystem: Andere functionaliteiten:
Delen van dezelfde data (zorgen dat simultaan lezen en simultaan schrijven niet tot inconsistenties zorgt. Cfr. hyperthreading in het vak "besturingssystemen".)
Beveiliging van data (beschermen tegen *ongeoorloofd gebruik* en *fysiek falen* van het DBS. *Fysiek falen*: databeveiliging dmv hammingcodes, CRC-codes, data-encryptie… Alsook automatisch backups maken van het ganse systeem.
Ongeoorloofd gebruik: alles goed beveiligd met paswoorden, geëncrypteerde verbindingen waar nodig, geen hackbare bugs…)
Architectuur: **drielagenarchitectuur**: Externe laag (met views) > Logische laag (relaties ed.) > interne laag (tabellen) > fysieke opslag (010011)

Mappings = het vertalen van commando's die bij het zien van een bovenliggende laag worden gegeven naar geschikte commando's voor de onderliggende laag.

Dataonafhankelijkheid: **fysieke dataonafhankelijkheid**
= het kunnen aanpassen van de interne beschrijving en de logische>interne mappings zonder dat de logische beschrijving dan de DB moet worden aangepast.

Dataonafhankelijkheid: **logische dataonafhankelijkheid**
= het kunnen aanpassen van de logische beschrijving en de externe>logische mappings zonder dat dit impact heeft op de views.

2. Hoofdstuk 2

Een **datamodel** is een verzameling van regels en voorschriften die het mogelijk maken om de structuur en het gedrag te beschrijven van de data. Een **databaseschema** is het schema dat data beschrijft die bij deze situatie relevant zijn.

Een **databasemodel** is een verzameling van regels en voorschriften 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.

Een **atomaire waarde** is een waarde (in een tabel) die niet verder kan opgesplitst worden. Voorbeelden van niet-atomaire-waarden: adres, datum, naam … (vaak samengestelde attributen).

!bekijken: fig 2.1 pag 33

(bekijk figuren van onderstaande, meer dan termen en indruk figuren moej ni weten)
Hiërarchisch model: boomstructuur, boomstructuur met virtuele 2e ouder
Netwerkmodel. (=graaf)
Structurele modellen: relationeel model
Semantische modellen: objectgeoriënteerd, objectrelationeel
Verdere ontwikkeling

3. Hoofdstuk 3

!Fig 3.1 pg 56: het volledige databaseontwerpproces: informatie-vergaring, conceptueel ontwerp, logisch ontwerp en fysiek ontwerp.

Een **CASE-tool** is een programma die helpt bij het ontwerpen van een (EER-, UML-)model of dit model automatisch opstelt aan de hand van broncode (bv DDL-scripts). CASE staat voor Computer Aided Software Engeneering.

EER staat voor Enhanced Entity Relationship. Het EER-model is een uitbreiding van het ER-model uit 1976.

Een **entiteit** is een 'ding' (voorwerp, gebeurtenis, dier, persoon…) dat een zelfstandig bestaan leidt in de reële wereld.
Een **entiteittype** karakteriseert een collectie van entiteiten en wordt gekenmerkt door een naam en een verzameling van attributen.

Een **relatietype** is een verwantschap tussen twee of meer entiteittypes. (Niet noodzakelijk verschillende). Een relatietype kan attributen hebben.

Enkelwaardige attributen/Meerwaardige attributen. Samengestelde attributen. Afgeleide attributen. Sleutelattributen. Deze vier zijn **orthogonaal** (kunnen naar willekeur gecombineerd worden).

Een **zwak entiteitstype** karakteriseert een collectie entiteiten die niet op zichzelf bestaan maar **bestaansafhankelijk** zijn van andere (**identificerende**) entiteiten. (bv een schilderij gemaakt door schilder, festival georganiseerd door organisator, examen van vak afgelegd door student…)

Relatierestricties: **kardinaliteitsrestricties** (0..N, 0..1, 0..N, 1..1, 1..N, N..N') en **participatierestricties** (totaal || partieel).

Een **subtype** is een entiteittype dat een subcollectie van entiteiten karakteriseert. (een deelverzameling van de entiteiten horend bij het overeenkomstig **supertype**)

Het creëren van specifiekere subtypes heet **specialisatie**, het creëren van algemenere supertypes **generalisatie**. Dit kan ondermeer via een attribuut gebeuren. (vb 'Type').

Karakteristieken en restricties: disjunct of overlappend, totaal of partieel.

Een **categorie** (of unietype) is een 'speciaal' subtype met verschillende supertypes, dat wordt ingevoerd om entiteiten te groeperen. (symbool: U van unie)

!fig 3.19 pg 80 : een n-air relatietype kan weergegeven worden dmv n identificerende binaire relatietypes. (en let op voor de connection trap!)

Optioneel: case-studies eens bekijken. (opstellen EER-diagram voor een DB)

4. Hoofdstuk 4

Een **atomair datatype** is een datatype waarop een algebra gedefinieerd is.

Het **schema** $R(A_1:T_1, A_2:T_2,\ldots,A_n:T_n)$ van een relatie is opgebouwd uit een relatienaam R en een eindige verzameling van attributen $\{A_1:T_1, A_2:T_2,\ldots,A_n:T_n\}$
Elk attribuut is opgebouwd uit een attribuutnaam A_i en een geassocieerd datatype T_i.

De **extentie** (=**corpus**) van een basisrelatie met schema $R(A_1:T_1, A_2:T_2,\ldots,A_n:T_n)$ is een met de tijd variërende verzameling van m n-tupels, die allemaal volledig worden gekarakteriseerd door de attributen uit het schema.
Het aantal attributen n noemen we de **graad** van een relatie.
Het aantal tuples m noemen we de **kardinaliteit** van de relatie.

Eigenschappen:
- Binnen een relatie kunnen geen dubbele tuples voorkomen
- De tupels van een relatie zijn niet geordend
- De attributen van een relatie zijn niet geordend
- Alle attribuutwaarden van een relatie zijn atomair

Naast data bevat het DBMS ook metadata. Dit is data voor interne werking en van geen belang voor diegene die interactie heeft met de DB.

Een **view** is een benoemde, virtuele relatie die is afgeleid van gebruikersgedefinieerde en/of systeemgedefinieerde basisrelaties en/of andere views. De definitie gebeurt door een **definiërende expressie**. *SQL: CREATE VIEW*

Waarom?
- Enkel genormaliseerde relaties moeten materialiseren in de DB
- Geen afleidbare data opnemen in basisrelaties
- Nieuwe data afleiden
- Bevragingsinstructies in te korten (‘gebruiksvriendelijker’)
- In individuele gebruikersbehoeften voorzien
- Data-afscherming

Zoeken: voor SQL geen onderscheid tussen views en basisrelaties. *SQL: SELECT FROM WHERE*

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 één (n+1)-tuple opgenomen in de lijst. Dit 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. Zodoende worden zoekopdrachten doorgaans enorm veel versneld, maar een index neemt ook veel geheugen in beslag. *SQL: CREATE INDEX ON*

SQL: NULL: goed nadenken welke attributen NOT NULL moeten zijn!
DEFAULT is ook mogelijk.

Als K een deelverzameling van de attributen van R is, dan is K een **kandidaatsleutel** als en slechts als:

Uniciteit: Geen enkele legale extentie van R bevat twee tuples met dezelfde waarden voor alle attributen van K

Irreducibel: als uit K attributen worden weggelaten geldt de uniciteit niet meer

Een artificieel ingevoerde kandidaatsleutel (omdat geen anderen voor handen zijn) noemen we aan **surrogaatsleutel**.

Elke relatie heeft één kandidaatsleutel als **primaire sleutel**. Best is de kandidaatsleutel met het kleinste aantal attributen te nemen.

Een **vreemde sleutel** is een attribuut die een referentie is naar een tuple van een andere relatie.
SQL: FOREIGN KEY REFERENCES
(kan ook door zelfreferentie)

Een **integriteitsrestrictie** is een voorwaarde waaraan alle data uit de database op elk moment moet voldoen. *SQL: CREATE ASSERTION (naam) CHECK // DROP ASSERTION (naam)*
We onderscheiden volgende cathegorieën van integriteitsrestricties: Toestands- vs. Transitierestricties en Relatie- vs. Databaserestricties.

Een **stored procedure** is een voorgecompileerde groep bewerkingen, geschreven in SQL/PSM (Persistent Storage Module) die de SQL-taal omvat. Voert een actie uit met DB of data en geeft evt een resultaat terug.

SQL: CREATE PROCEDURE naam(parameters)
lokale_declaraties
procedure_corpus

(oproepen met CALL-statement)

Relationele algebra: triviaal. *SQL:*
P RENAME R AS RE (R attribuut van relatie P)
UNION // INTERSECT // MINUS // TIMES // WHERE // JOIN // OUTER JOIN // DEVIDEBY // GROUP

5. Hoofdstuk 5

Bij een **logisch databaseontwerp** kiezen we een databasemodel. Bv: een (E)ER-model omzetten in een relationeel schema.

Hiervoor bestaan **omzettingsalgoritmes**. (E)ER->relationeel schema heeft zo een uitvoerig besproken (in het boek) algoritme van 9 stappen (gedragsspecificaties aanmaken exclusief), maar dit is eerder een vaardigheid dan blokwerk.

Een verzameling van attributen Y is **functioneel afhankelijk** van X als de waarden van Y op elk moment uniek worden vastgelegd door de waarden van X. Als de waarden van X bekend zijn, volgen die van Y daaruit. Daarom wordt X de **determinant** van deze functionele afhankelijkheid genoemd. We noteren X→Y
Y is **irreducibel functioneel afhankelijk** als X en Y geen gemeenschappelijke attributen hebben en uit X geen attributen kunnen worden weggelaten zodat Y functioneel afhankelijk blijft van X.
Y is **meerwaardig functioneel afhankelijk** van X (X→→Y) als de waarden van X een collectie mogelijke waarden van Y vastleggen.

Fig 5.15 pg 171: zo ziet een **functioneel afhankelijkheidsdiagram** er uit.

<u>NORMALISATIE</u>
Een relatie staat in **eerste normaalvorm** als de datatypes van de voorkomende attributen atomair zijn.
Een relatie staat in **tweede normaalvorm** als ze in eerste normaalvorm staat en elk attribuut van de relatie dat geen deel uitmaakt van een kandidaatsleutel irreducibel functioneel afhankelijk is van elke kandidaatsleutel van de relatie.
Een relatie staat in **derde normaalvorm** als ze in tweede normaalvorm staat en elk attribuut van de relatie dat geen deel uitmaakt van een kandidaatsleutel niet transitief irreducibel functioneel afhankelijk is van een kandidaatsleutel van de relatie.
Een relatie staat in **Boyce-Codd normaalvorm** als ze in eerste normaalvorm staat en elke determinant een kandidaatsleutel is van de relatie.
Een relatie staat in **vierde normaalvorm** als ze in Boyce-Codd normaalvorm staat en geen enkele meerwaardige functionele afhankelijkheid bevat, tenzij dit de enige afhankelijk is die voorkomt in de relatie.
We noteren **1NF**, **2NF**, **3NF**, **BCNF** en **4NF**.

Het afstappen van hogere vormen van normalisatie om betere prestaties (snelheid, opslag…) te verkrijgen heet **denormalisatie**.

6. Hoofdstuk 6

Tijdens de laatste ontwerpfase, het **fysieke databaseontwerp**, moeten het databaseschema, de (alternatieve) sleutels, de integriteitsrestricties, de stored procedures en de triggers worden geïmplementeerd in een gekozen (relationeel) dbms.

SQL = Structured Query Language, **DDL** = Data Definition Language en bestaat uit DDL-scripts. **DML** = Data Manipulation Language. SQL = DDL + DML.

SQLServer en **Oracle** bieden een uitbreiding van **SQL-99**-standaard aan. **MSAcces** ondersteunt deze standaard niet volledig. (Wat wou je anders van M$?)

BNF-notatie = Backus Naur Form (zie pagina 187)

Datadefinitietaal
CREATE TABLE naam **[AUTHORISATION** user**]**
DROP SCHEMA naam **[RESTRICT|CASCADE]**
CREATE DOMAIN naam **[AS]** datatype **[DEFAULT** waarde**] [CHECK** (expressie)**]**

CREATE TABLE ({kolomnaam datatype **[NOT NULL][UNIQUE][DEFAULT** waarde**] [CHECK**(expressie)**]**}[,…]{**PRIMARY KEY**(kolomnamen)[,]} ([**UNIQUE**(kolomnamen)] [,…]) ([**FOREIGN KEY**(kolomnamen) **REFERENCES** tabelnaam [**ON DELETE** actie] [**ON UPDATE** actie])[,…]) ([**CHECK**(expressie)][,…]))

DROP TABLE naam [**RESTRICT|CASCACE]**
CREATE [UNIQUE] INDEX naam **ON** tabel ((kolomnaam [**ASC|DESC]**[,…]))
DROP INDEX naam
CREATE VIEW viewnaam [(nieuwe_kolomnaam [,…]))] **AS** definiërende_expressie [**WITH CHECK OPTION**]
DROP VIEW naam
Zie pagina 188 en volgende.

Datamanipulatietaal
INSERT INTO naam [kolomlijst] **VALUES** lijst_met_attribuutwaarden
INSERT INTO naam [kolomlijst] select_instructie
LOAD FROM bestand **[DELIMITER** karakter**] INSERT INTO** naam [kolomlijst]
UPDATE naam **SET** ({kolomnaam=expressie}[,…])
DELETE FROM naam **WERE** expressie
INSERT INTO naam [kolomlijst] **VALUES** lijst_met_attribuutwaarden

SELECT [**ALL|DISTINCT]**{*|kolomexpressie [**AS** nieuwe_naam][,…])} **FROM** ({naam}[alias][,…])[**WHERE** conditie][{**GROUP BY** kolomlijst}[**HAVING** conditie]] [**ORDER BY** (kolomexpressie [ASC|DESC][,…])]
Zie pagina 195 en volgende

Wiskundig: **COUNT(*)**, **COUNT(DISTINCT,…)**, **COUNT(…)**, **MIN(…)**, **MAX(…)**, **SUM(…)**, **AVG(…)**, **EXISTS(…)**, **NOT EXISTS(…)**,

Stappen bij het uitvoeren van DML-instructies: fig 6.5 pg 208

Als een efficiënt uitvoerbare equivalente expressie gevonden is, zal het dbms een **queryplan** opbouwen.

Query-by-example: zie vb pg 212.

7. Hoofdstuk 7

In elke objectgeorieënteerde aanpak wordt gewerkt met **objecten** die een complexe waarde hebben, die de **toestand** van het object wordt genoemd. In een systeem dat werkt met datatypes worden objecten opgebouwd via **objecttypes**, waarin zowel de complexe datastructuur van de toestand, als de specifieke operaties die voor het object gedefinieerd zijn, door de softwareontwikkelaars zijn vastgelegd.

Slechts de signatuur van een operator zichtbaar maken heet het **inkapselen** van operatoren.

Overerving: Objectoriëntatie schrijft voor dat in nieuwe objecttypes een deel van de toetstands- en gedragsspecificaties van bestaande objecttypes overgenomen KAN worden. De toestand en het gedrag van het nieuwe objecttype worden dan bepaald door de combinatie van de overgenomen en de eigen specificaties. Dit mechanisme wordt het **overervingsmechanisme** genoemd.
Zo kunnen **netwerken** en **hiërarchieën** van overerving gevormd worden.

Operatoroverlading is overgeërfde operatoren herdefiniëren binnen het nieuwe objecttype en daarbij de naam van de operator behouden. Van het systeem zeggen we dat het (**operator**)**polymorfisme** ondersteunt.

Persistente objecten worden bewaard in permanent computergeheugen tot ze expliciet worden verwijderd, zo kunnen ze na afsluiten van het toepassingsprogramma toch blijven bestaan. Het **naamgevingsmechanisme** houdt in dat een object persistent kan worden gemaakt door het een persistente naam te geven (die uniek is de hele DB). Het **bereikbaarheidsmechanisme** zorgt ervoor dat elk object dat voorkomt (of naar verwezen wordt vanuit) een persistent object, ook persistent gemaakt wordt.

Elk object wordt geïdentificeerd door een unieke **ojectidentificator** (OID) die door het dbms wordt aangemaakt. Deze kan (ook door het dbms) niet worden aangepast. Na verwijdering van het object wordt het OID liefst niet meer hergebruikt.

Het **ODMG-model** (Object Database Management Group) is een pseudo-standaard.

Fig 7.1 pg223: **literaaltype** (structuur), **interface** (gedrag), **klasse** (structuur+gedrag)

Interfaces zijn bouwstenen die we gebruiken voor het modelleren van gedrag. Een interface is hoofdzakelijk opgebouwd uit operatoren.

Een klasse is de belangrijkste basisstructuur. Omdat zowel structuur als gedrag beschreven zijn is het object van een klasse volledig gedefinieerd.

Literaaltype:

Basistypes: (un)signed short/long, float, double, boolean, char en enum
Gestructureerde types: date, time, timestamp, interval (=een tijdsinterval) en zelfgemaakte structs
Collectietypes: In een collectie kunnen alleen maar data voorkomen van het opgegeven literaaltype. De belangrijkste soorten zijn de verzamelingtypes (set), bagtypes (bag), lijsttypes (list) en de rijtypes (array). In verzamelingen mag een element slechts éénmaal voorkomen, lists en arrays zijn geordend.

Bij **isa-overerving** moet het supertype een interface en het subtype een (andere) interface of een klasse zijn. Meervoudige overerving mag niet gecombineerd worden met operatoroverlading.

Bij **extends-overerving** moeten zowel supertype als subtype klassen zijn. Meervoudige overerving is niet toegestaan.

Een DA kan ervoor kiezen om het dbms voor een objecttype een benoemde, persistente verzameling te laten bijhouden die bestaat uit alle persistente objecten van het objecttype. Deze verzameling heet de **extentie** van het objecttype.

ODL=ObjectDefinitionLanguage. **OQL=**ObjectQueryLanguage. Elke keer we met een OQL-instructie werken met de elementen uit een collectie, moeten we een **iteratorvariabele** gebruiken.

Met de **SQL3**-standaard hebben ook relationele databases faciliteiten voor objectorieëntatie (dit is eigenlijk een **objectrelationeel model**).

OQL: **REF IS** id {**DERIVED|SYSTEM GENEREATED**} // voor **tuple-identificator**
OF attribuutnaam **REF**(naam_tupletype)[**SCOPE**(tabelnaam)] voor extra attribuut

De specificatie van overerving vindt plaats met het sleutelwoord **UNDER** in SQL3

Multimedia-objecten: clob (character large objects) en **blob** (binary large objects)
XML = Extensible Markup Language

De omzetting van objecttypes naar een relationele database heet **objectrelationele mappings**. (bv ‘Hibernate’, open source software)

8. Hoofdstuk 8

Een **API** (**applicatieprogramma-interface**) voor databasetoegang zorgt voor de verbinding of interface tussen de applicatie en het dbms.
Er bestaan ook **ingebouwde API's**.
Hoofdtaken van een API: opzetten van dbms-connectie, selecteren van database(component), doorgeven van instructie, ophalen van resultaat, afsluiten van dbms-connectie. (cfr PHP-code die steeds deze structuur heeft.)

Embedded SQL is één van de eerste gebruikte technieken om relationele databases te bereiken vanuit applicatiecode. Embedded SQL stelt je in staat om SQL-instructies via specifieke taalconstructies in te bouwen in de programmacode.

Ingebedde SQL-instructies beginnen met EXEC SQL en eindigen met een puntkomma.

Embedded SQL voorziet een constructie om in een itaratorconstructie de tuples één voor één op te halen en te verwerken. Dit heet een **cursor**.

Statische SQL / **Dynamische SQL** (=SQL met **late binding**)

SQLJ = Embedded-SQL tegenhanger voor Java.

Call-Level API: apparte API die tussen app en dbms zit, zie fig 8.5 pg 255

De standaard **ODBC** (Open DataBase Connectivity) stelt je in staat om relationele databases (of andere databronnen die werken met tabellen) op een dbms-onafhankelijke wijze te benaderen vanuit applicaties. **JDBC** is opnieuw de Java-tegenhanger.

SQL/CLI (call level interface) is een deel van de SQL-standaard en dus ook bedoeld voor toegang tot relationele databases. De technologie bouwt verder op ODBC, waardoor er veel overeenkomsten zijn.

OLEDB (Object Linking and Embedding for DataBases) werd ontwikkeld door M$ om heterogene databronnen op een uniforme wijze vanuit een applicatie te kunnen benaderen en delen. Door de nauwe verwantschap met C++ is OLEDB moeilijk rechtstreeks aan te roepen vanuit applicaties die geschreven zijn in een programmeertaal die geen pointers ondersteunt. Om dit euvel te verhelpen werd **ADO** (ActiveX Data Objects) als extra toegangslaag voor applicaties toegevoegd aan OLEDB. ADO is gebruiksvriendelijker.

Databasetoegang via webpagina's: **ASP** (Active Server Pages), **JSP** (Java Server Pages) en **PHP** (PHP : Hypertext Preprocessor, vroeger Personal Home Page)

Een aantal call-level API's kunnen ook worden gebruikt vanuit frameworks zoals **J2EE** (Java 2 platform Enterprise Edition) en **.NET**-framework.

9. Hoofdstuk 9

De grootste gevaren bij ongeoorloofd gebruik:
- Diefstal / fraude
- Schending van de privacy
- Beschadiging van de database
- Sabotage van de database

Uitgebreider: fig 9.1 pg 269

Beveiligingsstrategiëen:
- Gebruikers: acties registreren in een logbestand (aka **auditbestand**),
  - Waarvoor **toegangscontrole** noodzakelijk is.
  - Bevoegdheden verdelen over versch. personen.
- Hardware: verhinderen diefstal en beschadiging hardware alsook bijplaatsen
- Software: ingebouwde beveiliging OS en dmbs optimaal benutten, geen gebruik maken van default paswoorden, software updaten, beheerders die bv ontslagen zijn verwijderen, toegangsprogramma's ook controleren op bugs.
- Data: **toegangsbeperkingen** voor gebruikers. Datacommunicatie **versleutelen**.

Dit alles samengevat in fig 9.2 pg 272.

Authentificatie is het controleproces om uit te maken of een gebruiker is wij hij of zij beweert te zijn. *SQL: CREATE USER naam IDENTIFIED BY paswoord // ALTER USER naam IDENTIFIED BY nieuw_paswoord // DROP USER naam*

Een instructie waarvoor expliciet toestemming wordt verleend wordt een **privilege** genoemd. Privileges kunnen ook **contextafhankelijk** zijn (cfr WHEN).
SQL: GRANT {privilegelijst|ALL PRIVILEGES} ON {componentnaam | [databasenaam.] | *.*} WHEN expressie TO {gebruikersnamen|PUBLIC} {WITH GRANT OPTION} // REVOKE [GRANT OPTION FOR] {privilegelijst | ALL PRIVILEGES} ON {componentnaam | [databasenaam.]* | *.*} FROM {gebruikersnamen|PUBLIC} {RESTRICT|CASACE}*

Een **gebruikersprofiel** is een benoemde verzameling van privileges, die als geheel aan een gebruiker kunnen worden toegekend. *SQL: CREATE ROLE naam // DROP ROLE naam*

Een **beveiligingsniveau** kan worden gezien als een label dat aangeeft hoe goed een bepaald gegeven moet worden afgeschermd. Een **toegangsniveau** wordt gezien als een label die aangeeft tot welke beveiligingsniveau's een gebruiker toegang heeft.
Lezen: toegangsniveau gebruiker >= beveiligingsniveau data.
Schrijven: toegangsniveau gebruiker <= toegangsniveau data. **= Stereigenschap**
Een toegangscontrole die werkt met beveiligingsniveau's wordt in de literatuur ook **MAC-toegangscontrole** genoemd. (Mandatory Access Control)

De **meer-niveau-relatie** is een relatie waarbij voor elk attribuut een toegangsniveau bijgehouden wordt.

In sommige gevalen kunnen de tuples voor een lager toegangsniveau worden afgeleid uit de tuples voor een hoger toegangsniveau. Dit heet **filteren**. In andere gevallen kunnen de data niet worden verkregen door filteren en zijn er extra tuples nodig om het verschil in attribuutwaarden op verschillende toegangsniveaus te kunnen weergeven. Dit noemen we **polyinstantiatie**.

Er bestaan manieren om een **tracker** (zoekconditie voor een **convert-channel**) uit te schakelen. Één ervan is om het maximaal aantal zoekconditities te beperken. Dit heet **query restriction**. Ook kunnen de attribuutwaarden in de database gepermuteerd of verplaatst worden, dit heet **data swapping**.

Een auditbestand kan de volgende informatie bevatten:
- Gebruikersnaam van de gebruiker
- Instructie gegeven door gebruiker
- Datum en tijdstip van instructie
- De locatie van waaruit de instructie werd uitgevoerd
- De impact van de instructie:
  - -waarop het inwerkte
  - -de toestand ervan voor de instructie (zogeheten **before-image**)
  - -de toestand ervan na de instructie (zogeheten **after-image**)

Versleuteling: assymetrische versleuteling(salgorimen) en **symmetrische versleuteling(salgoritmen)**. Triviaal voor iemand met wiskundige achtergrond :p

Beveiliging kan ook gebeuren door de gebruikers enkel views aan te bieden waardoor deze evenmin rechtstreeks aan de data kan.

10. Hoofdstuk 10

Een **transactie** is een hoeveelheid werk (sequentie van instructies) waarvan het dbms garandeert dat het ofwel volledig met succes wordt uitgevoerd, ofwel helemaal niet. Na een geslaagde transactie volgt een **COMMIT**-instructie, anders is er sprake van een **ROLL-BACK**-instructie. Omdat deze taak enorm belangrijk en vaak zeer complex is, hebben de meeste dbms's een speciale component die er voor instaat. Deze heet de **transactiemanager** (transaction processing monitor). *SQL: START TRANSACTION … COMMIT WORK … ROLLBACK WORK … RETURN;*

Alle transacties moeten voldoen aan de **ACID-eigenschappen**, zijnde
- Atomair (alles of niets qua uitvoering)
- Consistent (consitentie van de database bewaren)
- Isolatie (dbms moet garanderen dat transacties onderling geïsoleerd worden uitgevoerd)
- Duurzaam (resultaten gaan niet verloren, zelfs bij onmiddellijk nadien falen)

Om bij te houden waar expliciete transacties beginnen en te weten tot waar eventuele ROLLBACK-instructies werk ongedaan moeten maken, werkt een dbms met **synchronisatiepunten**. (cfr fig 10.3 pg 299)

OLTP = Online Transaction Processing

Sommige dbms's laten toe dat gebruikers zelf tijdens de transactie een soort van tussentijds synchronisatiepunt kunnen plaatsen. Deze worden **savepoints** genoemd.

Oorzaken van falen:
- Overmacht
- Beschadiging hardware
- Softwarefouten (bugs)
- Bewuste onderbreking
- Onachtzaamheden v/d gebruikers (verkeerde instructies op db)
- Sabotage of diefstal bij ongeoorloofd gebruik

Soft crash: enkel de buffers (hoofdzakelijk RAM) gaan verloren. **Hard crash**: ook data in het secundaire geheugen (vnl harde schijven) gaat verloren.

Hoe voorkomen? **Backups**. **Raid (1, 5** of **15)**. **Logbestand** met wijzigingen sinds laatste backup.

Databasebuffers ledigen heet **flushen**. (zoals bij alle buffers en logs derpa herp)

Met **steal** wordt bedoeld dat het dbms een flush mag uitvoeren terwijl er nog transacties niet commited zijn. Met **no force** wordt bedoeld dat flushing niet mag worden veroorzaakt door een impliciete of expliciete COMMIT. Meeste dbms's zijn steal-no force. **No steal** en **force** spreken voor zich.

De **write-ahead-log regel** is een maatregel die stelt dat COMMIT van een transactie naar het logbestand moet worden geschreven net voor de transactie wordt afgesloten.

Een **controlepunt** is een tijdstip waarop de database en het logbestand worden gesynchroniseerd door alle databasebuffers te flushen.

Verwerken van transacties door dbms en herstel van soft crashen dmv controlepunen: evt eens doorlezen. Pg 309-310.

Een alternatief voor hersteltechnieken die gebruikmaken van een logbestand is het werken met **schaduwpagina's**. Bij deze hersteltechnieken deelt met de pagina's in het magneetschijfgeheugen op in werkpagina's en schaduwpagina's zodat elke werkpagina een bijhorende schaduwpagina heeft. Door deze techniek kunnen naast de soft crashes ook de hard crashes (met uitzondering van falen van te veel harde schijven) worden hersteld.

11. Hoofdstuk 11

Parallelle verwerking van transacties vs **interleaved**.
Interleaved verwerking heeft natuurlijk dezelfde problemen als alle multithreaded software… (heten hier **lost update**, **uncommited dependency** en **inconsistente analyse**)
Ook doen problemen zich voor wanneer een transactie dezelfde gegevens meermaals opvraagt doch verschillende waarden krijgt omdat deze tussentijds gewijzigd worden door andere transacties. Dit heet een **nonrepeatable read**. Ook kan een **extra** record (**phantom**) tussentijds worden toegevoegd, dit heet **phanom read**.

Als een instructieset van transacties via omvormingsregels kan worden omgezet naar een seriële sequentie zegt men dat de instructiesequentie **serialiseerbaar** is. De instructiesequentie heeft dan dezelfde resultaten als de verkregen seriële sequentie. Omvormingsregels aangaande welke data gelezen en/of geschreven wordt, noemen we de **conflictserialiteitsregels**.

Isolatieniveau's in SQL:

- **Read uncommited**: minst veilig. Er kunnen dingen gelezen worden die al aangepast maar nog niet COMMITED zijn
- **Read commited**: Geen dirty reading. Repeatable read en phantom read blijven mogelijk
- **Repeatable read**: phantom read blijft mogelijk, dirty- en repeatable read niet
- **Serializable**: speelt volledig op safe.

Een **timestamp** is een unieke indicator die de relatieve starttijd van een transactie aanduidt. Om te kunnen aanduiden wanneer een gegeven voor het laatst gelezen of aangepast werd, moet het dbms een **lees-timestamp** en een **schrijf-timestamp** kunnen koppelen. Bij timestamping-methoden is het belangrijk te weten hoe groot de data-eenheid is waaraan de timestamp wordt gekoppeld, dit noemen we de **granulariteit** van de methode.

Een **lock** is een soort van reservering op een gegeven, waardoor een transactie een exclusieve of gedeelde toegang krijgt tot het gegeven (→ **exclusieve lock** en **gedeelde lock**'s). Als een lock niet kan worden verkregen zet het dbms de transactie in wachttoestand. Het **twee-fase locking protocol** houdt in dat er in de eerste fase, de **groeifase**, locks kunnen aangevraagd (en verkregen) worden en er in de tweede fase, de **krimpfase**, locks kunnen worden vrijgegeven. Maar niet andersom. Dit leidt tot het **twee-fasen locking theorema**: *Als alle transacties voldoen aan het twee-fasen-locking protocol dan zijn de instructiesequenties van alle gelijktijdige uitvoeringen van deze transacties serialiseerbaar.*

Deadlock-preventie: **wait-die**-techniek (enkel oudere transacties kunnen wachten op jongere) en **wound-wait**-techniek (enkel jongere kunnen wachten op oudere)

Preventief ingrijpen om concurrency-problemen te vermijden heet een **pessimistische methode**. Enkel ingrijpen in geval van een concurrency-probleem heet de **optimistische methode**. Dat laatste is aangeraden in omgevingen waarin zich zelden problemen voordoen (CPU-overhead vs rollback-overhead).