Samenvatting Databanken II

TIN 2 - HoGent

Lorenz Verschingel

6 mei 2015

1 SQL

1.1 Inner join

Voorbeeld van een inner join:

```
SELECT au_lastName, au_FirstName, title_id
FROM authors
JOIN titleAuthor ON authors.au_id = titleauthor.au_id
```

Hier worden alle records van authors en titleAuthor aan elkaar gekoppeld op basis van au_id.

1.2 Aliassen

Het gebruik van tabel aliassen gebeurt via het keyword 'AS' of door een spatie.

```
SELECT au_lname, au_fname, title_id
FROM authors AS A
JOIN titleauthor TA ON A.au_id = TA.au_id
```

1.3 Inner join van meerdere tabellen

Gegevens kunnen ook over meerdere tabellen verspreid zitten. Hierbij moeten dan meerdere tabellen aan elkaar gekoppeld worden.

```
SELECT au_lname, au_fname, title
FROM authors A
JOIN titleauthor TA ON A.au_id= TA.au_id
JOIN titles T ON TA.title_id= T.title_id
```

Het kan zijn dat er enkel gegevens uit 2 tabellen worden getoond, maar dat er in werkelijkheid meerdere tabellen gekoppeld zijn omdat het geen directe koppeling is tussen de tabellen waaruit de gegevens komen.

1.4 Outer join

Een outer join retourneert alle records van 1 tabel, zelfs als er geen gerelateerd record bestaat in de andere tabel.

Er zijn 3 types van outer join:

1. Left outer join retourneert alle rijen van de eerst genoemde tabel in de FROM clause.

   In sql is dit de LEFT JOIN

2. Right outer join retourneert alle rijen van de tweede genoemde tabel in de FROM clause.

   In sql is dit de RIGHT JOIN

3. Full outer join retourneert ook rijen uit de eerste en tweede tabel die geen corresponderende entry hebben in de andere tabel.

   In sql is dit de CROSS JOIN

1.5 Union

Via een UNION combineer je het resultaat van 2 of meerdere queriesin 1 resultaattabel.

```
SELECT ... FROM ... WHERE ...
UNION
SELECT ... FROM ... WHERE ...
ORDER BY ...
```

Regels:

- De resultaten van de 2 SELECT opdrachten moeten evenveel kolommen bevatten.
- Overeenkomstige kolommen uit beide SELECT'smoeten van hetzelfde data type zijn en beide NOT NULL toelaten of niet.
- Kolommen komen voor in dezelfde volgorde
- De kolomnamen/titels van de UNION zijn deze van de eerste SELECT
- Het resultaat bevat echter steeds alleen unieke rijen
- Aan het einde van de UNION kan je een ORDER BY toevoegen. In deze clausule mag geen kolomnaam of uitdrukking voorkomen indien kolomnamen van beide select'sverschillen. Gebruik in dat geval kolomnummers.

1.6 Subqueries

Bij een subquerykomt een selectie voor als onderdeel van een andere selectie.

```
SELECT ...
FROM
WHERE voorwaarde
```

De voorwaarde bevat in het rechterlid tussen ronde haakjes een nieuwe SELECT.

De outer level query is de eerste select. Deze bevat de hoofdvraag.

De inner level query is de tweede select deze staat in de WHERE of HAVING clause.

We gebruiken subqueries om:

- een resultaat te retourneren waarbij de subquery een proces gegeven bevat.
- gegevens uit meerdere tabellen te halen. Dit kan vergeleken worden met een JOIN. Enkel worden bij subqueries de tabellen afzonderlijk gebruikt.

Er zijn drie vormen in de WHERE clause

1. Geneste subvragen
2. Gecorreleerde subvragen
3. Operator exists

Subqueries kunnen ook voorkomen in de FROM en SELECT clause.

1.6.1 Geneste subvragen

De subvragen worden altijd eerst uitgevoerd en moeten steeds tussen haakjes staan. Subvragen kunnen in meerdere niveau's genest zijn.

Bij een geneste subquery kan de één waarde geretourneerd worden of een ganse lijst met waarden.

ANY en ALL keywords worden gebruikt in combinatie met de relationele operatoren en subqueriesdie een kolom van waarden retourneren.

- ALL retourneert TRUE als alle waarden geretourneerd in de subquer yvoldoenaande voorwaarde.
- ANY retourneert TRUE als minstens 1 waarde geretourneerd in de subquery voldoet aan de voorwaarde.

1.6.2 Gecorreleerde subqueries

Bij een gecorreleerde subquery hangt de inner query af van informatie van de outer query. Voor elke rij uit de hoofdvraag wordt de subvraag opnieuw uitgevoerd. Bijgevolg gebruikt me beter JOIN als dit mogelijk is.

```
SELECT ...
FROM tabel a
WHERE uitdrukking operator (
        SELECT ...
        FROM tabel
        WHERE uitdrukking operator a.kolomnaam)
```

In de hoofdvraag mag je geen velden gebruiken uit de subvraag, maar wel omgekeerd.

1.6.3 Exists operator

Via de operator EXISTS wordt getest op het al dan niet leeg zijn van een resultaatset. Er bestaat ook NOT EXISTS.

2 Database architecturen

2.1 Componenten van een database

2.1.1 Hardware

- PC
- Mainframe
- Netwerk van computers ← multi-user

2.1.2 Software

- DBMS
- Applicaties
- Netwerk-software
- Operating System

2.1.3 Data

- Brug tussen machine en mens
- Operationele data → database
- Metadata → systeem cataloog

2.1.4 Procedures

- Hoe het DB-systeem te gebruiken
- Inloggen
- Hoe een specifiek BD-programma gebruiken
- ...

2.1.5 Mensen

- DB admins
- DB ontwerpers
- Applicatie-ontwikkelaars
- Eindgebruikers

2.2 Multi-user DB

Bij multi-user DB's gebuiken verschillende gebruikers tegelijk de data in de DB. Wanneer verschillende transacties enkel data lezen is er geen probleem. Als minstens één transactie data wijzigt kunnen wel problemen optreden.

⇒ Concurrency control: voorkomen van problemen.

⇒ Recovery: herstel in geval van problemen.

2.3 Gedistribueerde DB

Bij een gedistribueerde DB bevinden de data (en metadata) zich fysisch op verschillende plaatsen.

Gedistribueerde DBMS nodig: een logische DB die verdeeld is in fragmenten. Elk fragment is gestockeerd op één of meerdere computers.

Elke site kan autonoom beslissen over zijn eigen data.

Wanneer data nodig is uit een andere site → global application.

2.4 Data warehouse

Geïntegreerde view van data uit heteregone data sources om beslissingen van beleidsmakers te ondersteunen.

Er wordt enkel gelezen (niet weggeschreven).

Moet snel ingewikkelde queries kunnen uitvoeren.

2.5 Hardware architectuur

Data-intensieve applicaties bevatten 5 hoofdcomponenten:

1. database
2. DBMS
3. dataprocessing-logica
4. business-logica
5. user interface

2.5.1 Teleprocessing

Éen centrale processor verbonden met een aantal terminals. Zie figuur 1a.

2.5.2 File-Server

Éen file-server verbonden via LAN met workstations. Zie figuur 1b.

2.5.3 2-tier client-server (Fat client)

Client via een netwerk verbonden met een server. Zie figuur 1c

2.5.4 3-tier client-server

Er is een bijkomende middle tier. Zie figuur 1d. Op deze figuur is de client tier verantwoordelijk voor de presentatie, de middle tier voor de logica en de database tier voor de data.

2.5.5 n-tier client- server

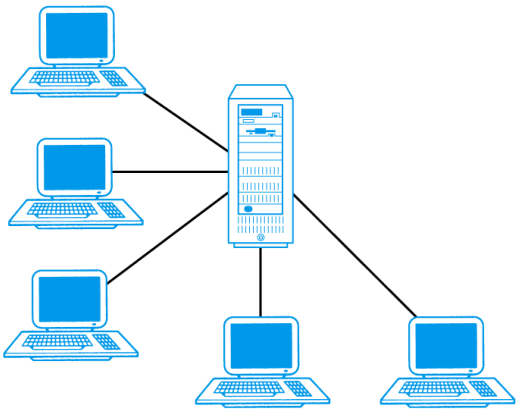
De middle tier wordt opgesplitst voor meer flexibiliteit, schaalbaarheid, beschikbaarheid...

2.5.6 Applicatieserver

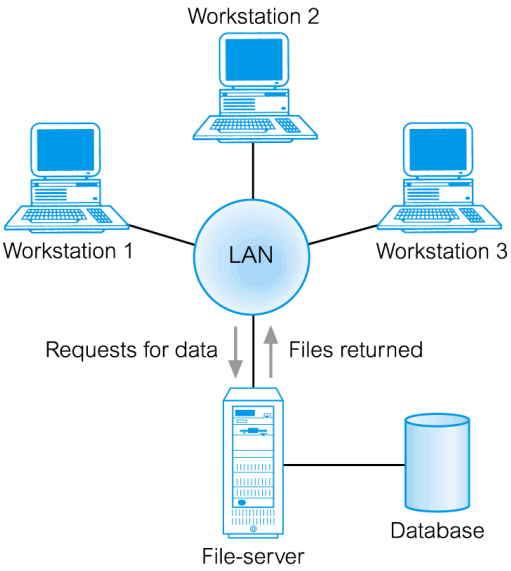
Een server waarop applicaties worden aangeboden aan de client.

De clients kunnen de aangeboden applicaties op hun eigen werkplek bedienen, terwijl onderhoud, beheer en ondersteuning op afstand plaatsvinden (op de server).

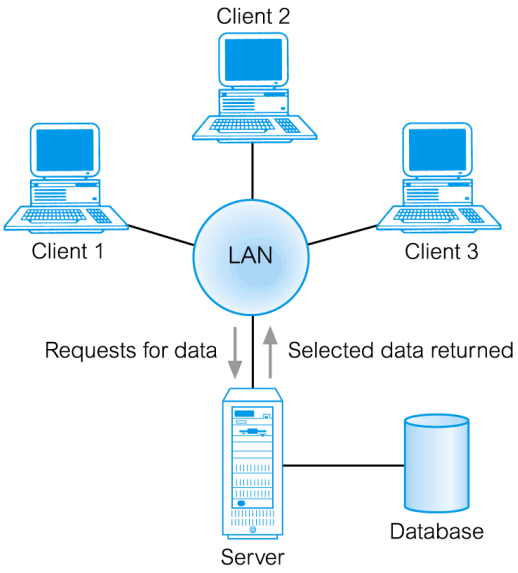
Zorgt ervoor dat via een application programming interface (API) business logica en business processen kunnen gebruikt worden daar andere applicaties.



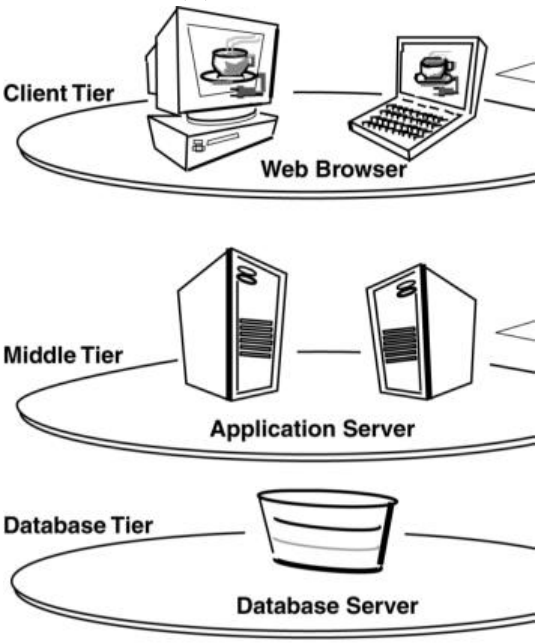
(a) Teleprocessing



(b) File-Server



(c) 2-tier client-server



(d) 3-tier client-server

Figuur 1: Verschillende harware-architecturen

2.6 Middleware

Middleware is software die softwarecomponenten of applicaties verbindt. Dit is nodig wanner gedistribueerde systemen te complex worden zonder een gemeenschappelijke interface.

⇒ Doel: verbergen van onderliggende complexiteit van gedistribueerde systemen.

2.6.1 Asynchrone Remote Procedure Call

Bij asynchrone RCP stuurt de client een verzoek naar de server zonder te wachten op antwoord. Als de connectie verbroken wordt dan moet de client opnieuw beginnen.

⇒ Gebruiken indien integriteit niet belangrijk is.

Er zijn zes types middleware:

1. Asynchrone RPC
2. Synchrone RPC
3. Publish/Subscribe
4. Message oriented middleware
5. Object Request Broker
6. SQL-oriented data acces

2.6.2 Synchrone Remote Procedure Call

Bij synchrone RCP stuurt de client een verzoek naar de server wacht op antwoord. Zolang de server bezig is met het verzoek is de client geblokkeerd.

⇒ Gebruiken indien integriteit uiterst belangrijk is.

2.6.3 Publish/Subscribe

Publish/Subscribe is een asynchroon messaging protocal. De clients zijn abonnees en kunnen zich inschrijven om boodschappen van publishers te ontvangen.

Boodschappen zijn gecatalogeerd in klassen en een abonnee kan zich inschrijven in één of meerdere klassen.

2.6.4 Message Oriented Middleware

Bij MOM staat de software zowel op de client als op de server.

Er gebeuren asynchrone calls tussen client- en server-applicaties.

Er is een wachtrij die dient als tijdelijke opslag indien de bestemming bezet of niet geconnecteerd is.

2.6.5 Object Request Broker

Bij ORB is er communicatie en data-uitwisseling tussen objecten.

Common ORB Architecture (CORBA) is een standaard om software componenten, geschreven in verschillende talen en draaied op verschillende computers, met elkaat te laten communiceren.

2.6.6 SQL-oriented data access

SQL-oriented data access verbint applicaties met een database over het net. Het vertaalt SQL requests naar de eigen SQL van de database.

2.6.7 Middleware voor transactiebeheer

Transaction Processing Monitor

Complexe applicaties zijn vaak gebouwd op verschillende resource managers.

TP Monitor controleert het dataverkeer tussen clients en servers.

TP Monitor is een middleware component die toegang geeft tot de diensten van een aantal resource managers en zorgt voor een uniforme interface voor programmeurs die transactionele software ontwikkelen.

Er is een verhoogde *schaalbaarheid* door transaction routing.

Er is *taaklastbeheersing* want de TP monitor kan zien welke server het minst belast is en daar de client-calls naar toe dirigeren.

Er zijn *hetrogene bronnen* mogelijk.

Er is *funneling*: niet alle clients moeten constant online zijn $\rightarrow$ TP monitor kan user requests sturen naar reeds openstaande connecties.

Kortweg kan gezegd worden dat de TP monitor een transaction manager is.

2.7 Webservices

Een webservice is een software systeem, dat ontworpen werd voor interactie tussen applicaties en webservers.

Webservices delen business logica, dataen processen m.b.v. een programmeerbare interface over een netwerk.

Ontwikkelaars voegen de webservice toe aan een webpagina om specifieke functionaliteit aan te bieden aan de gebruikers.

Webservices gebruiker algemeen aanvaarde technologieën en standaarden zoals: XML, SOAP, WSDL en UDDI.

2.8 Service-Oriented Architecturen

Business-georiënteerde software-architectuur voor het bouwen van applicaties en business processen implementeren als een set van gepubliceerde services.

Granulariteit moet relevant zijn voor de service-gebruiker.

Services kunnen aangeroepen, gepubliceerd en gevonden worden op een abstracte manier, d.w.z. los van de implementatiedetails, maar gebaseerd op een unieke standaar-interface.

Laat toe dat applicaties zich snel kunnen aanpassen aan wijzigende business-processen t.g.v. bijvoorbeeld gewijzigde marktomstandigheden.

2.9 Cloud Computing

2.9.1 Kenmerken

On-demand self-service: Gebruikers kunnen clouddiensten verkrijgen, configureren en deployen zonder hulp van een provider.

Broad network acces: Toegankelijk van overal, vanaf elke standaardplatform.

Resource pooling: Computer resources van provider zitten in een pool, die meerdere gebruikers bedient. Fysieke en virtuele resources worden dynamisch toegekend volgens de behoefte van een gebruiker op een bepaald moment.

Snelle elasticiteit: Piekbehoeften van de klant worden opgevangen door pooling. Er is een sterk verminderd risico op uitval en onderbrekingen van de dienstverlening. Toekenning van de capaciteit kan geautomatiseerd gebeuren op basis van vraag → *scalability*.

Metingen van afgenomen diensten: Provider meet gebruik van storage, CPU, bandbreedte... Deze metingen worden dan gebruik voor de Service Level Agrements (SLA) en de facturatie.

2.9.2 Software as a Service

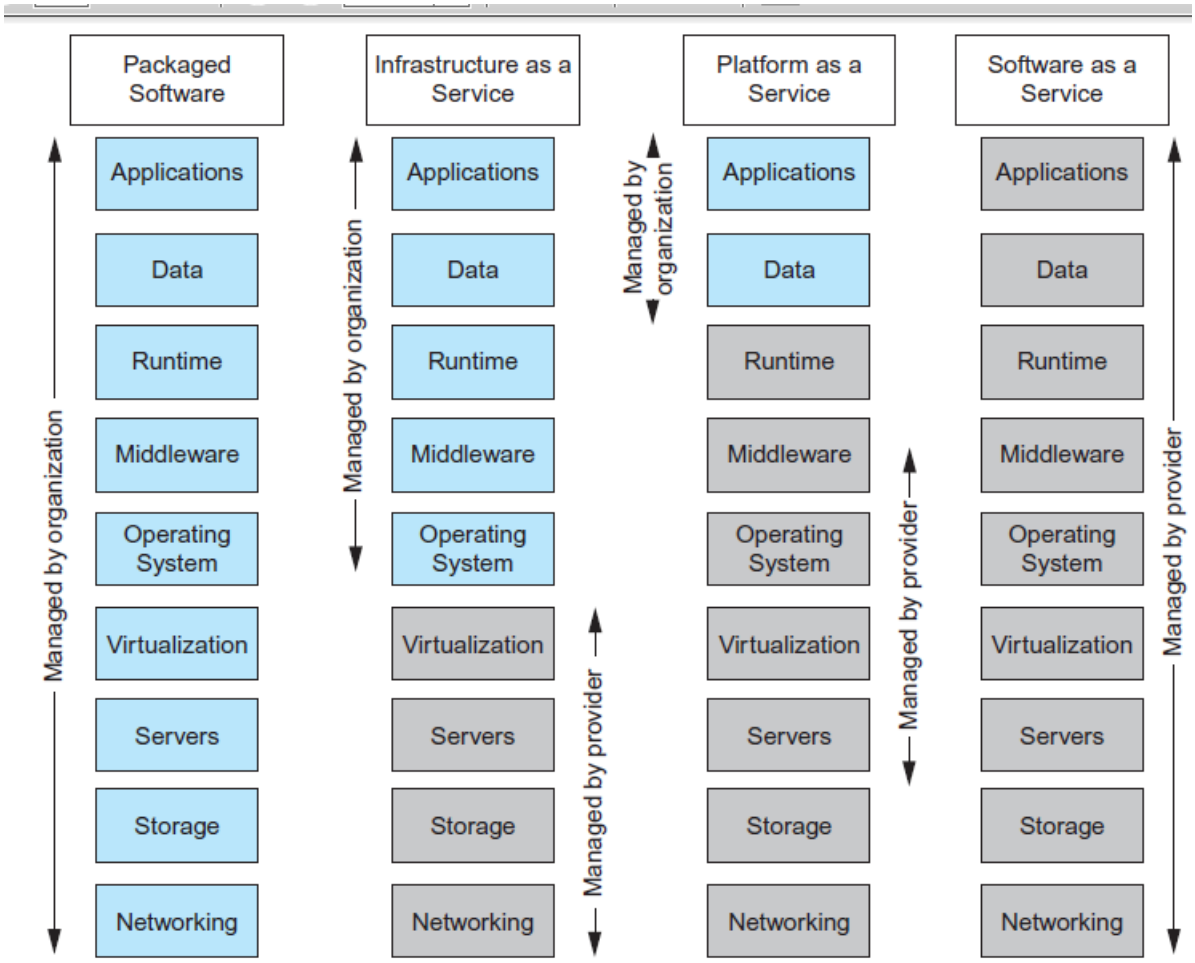
Bij SaaS bevinden zich software en data in de cloud. De toegang wordt verzorgt via een thin client (bv. web browser). De gebruiker kan hier beperkte toegang hebben tot configuratie-instellingen.

2.9.3 Platform as a Service

Bij PaaS bouwt de klant de webapplicatie zonder aankoop/onderhoud van software en infrastuctuur. De provider beheert de infrastructuur (netwerk, OS en storage). De klant beheert het deployment en de configuratie van applicaties.

2.9.4 Infrastructure as a Service

Bij SaaS levert de provider servers, storage, netwerk en OS aan de gebruikers, gebundeld al on-demand service. Dit model wordt typisch gebruikt voor platform-virtualisatie. De facturatie gebeurt volgens gebruik.



Figuur 2: Vergelijking van service-modellen

2.9.5 Voordelen van cloud computing

Kostenreductie: vermijd kapitaalintensieve investeringen voor opbrengsten.

Scalability: bijkomende resources schaf je aan volgens behoefte.

Security en betrouwbaarheid: providers kunnen expertise en resources verdelen over meerdere klanten. Een individuele klant kan dit niet betalen.

Toegang tot nieuwste technologieën: systemen moeten niet meer afgeschreven zijn om nieuwe technologieën te kunnen gebruiken, want je huurt de systemen.

Snellere ontwikkeling: platform van de provider zorgt voor software en diensten die ontwikkelingscyclus kunnen versnellen.

Prototyping/Load testing op grote schaal: de provider heeft hiervoor de resources.

Verhoogde competitiviteit: organisaties kunnen focussen op hun kerncompetenties i.p.v. op IT-infrastructuren.

Flexibeler werken : bv. toegang via mobile devices.

2.9.6 Risico's van cloud computing

Afhankelijkheid van het netwerk: Stroomonderbrekingen, bandbreedte-problemen en onderbreking dienstverlening.

Afhankelijkheid van de systemen: je vertrouwt op beschikbaarheid en betrouwbaarheid van systemen van de provider.

Afhankelijkheid van de cloud provider: deze kan failliet gaan of overgenomen worden door concurrent met mogelijk onmiddelijke stopzetting van de service.

Gebrek aan controle: je beslist niet zelf over bv. backup-policy.

Gebrek aan transparantie over achterliggende verwerking

2.10 Cloud gebaseerde databanken

Cloud gebaseerde databanken zijn een voorbeel van SaaS. Er zijn twee basiscategorieën:

1. Data as a Service (DaaS)
2. Database as a Service (DBaaS)

Het verschil tussen beiden zit hoofdzakelijk in hoe de data beheerd wordt.

2.10.1 Database as a Service

DBaaS biedt een volledige databankfunctionaliteit aan ontwikkelaars. Dit model voorziet ook in een management-laag, die continue monitoring en cofiguratie van de databank toelaat:

- geoptimaliseerde scalability
- hoge beschikbaarheid

- multi-tenancy: meerdere client-organisaites worden gelijktijdig bediend.
- effectief reource management in de cloud → ontwikkelaar moet zich niet bezighouden met dba-taken.

2.10.2 Data as a Service

Bij DaaS zit de data-definitie en querying in de cloud.

DaaS implementeert geen typische DMBS-interface (bv SQL), maar de data is beschikbaar via API's. → verhoogde controle door de provider.

DaaS laat toe om je data aan anderen ter beschikking te stellen.

2.11 Mobile Databases

Mobile databases dienen om data persistent te maken. Hierbij zijn er twee mogelijkheden waarbij de keuze gemaakt wordt door de app developer.

1. Database op afstand die verbonden wordt met het mobiele toestel (internet nodig).
2. Database die opgeslagen wordt op het mobiele toestel (offline gebruik mogelijk).

Vaak wordt met een middle tier zoals een mobile web service gewerkt. Hierbij is er geen rechtstreeks contact tussen de client en de databank.

De client stuurt een HTTP request naar de webservice. De client kan vaak in zijn request naar de webservice kiezen hoe hij de data wil ontvangen (=content negotiation). De webservice communiceert met de databank en haalt de data binnen. Naar gelang wat werd gevraagd, zal de web service de resultset omzetten. Dit resultaat wordt dan aangeboden aan de client. Op die manier is er nooit rechtstreeks contact tussen de databank en de client.

2.11.1 Voordelen

Er is nooit rechtstreeks contact tussen de client en de DB.

Clients kunnen kiezen in welk formaat ze de data wensen te ontvangen.

Schaalbaarheid hangt niet enkel af van de databank maar ook van de webservice → wordt verdeeld.

Dezelfde web service (logica) kan gebruikt worden door verschillende clients/platformen → reuse.

Request kan over https gebeuren → veiligheid.

2.11.2 Nadelen

Performantie wordt o.a. bepaald door de internetverbinding.

Behalve een DB Server ook nog nood aan eene extra technologie voor de middle tier.

Er is steeds internetverbinding nodig.

2.11.3 Mobile database lokaal

De lokale mobile database wordt ook wel de offline storage genoemd.

De DB is platform afhankelijk, maar is wel relatief snel.

De DB neemt opslagplaats in op het device. Hierdoor is de app zelf verantwoordelijk voor het updaten van de data.

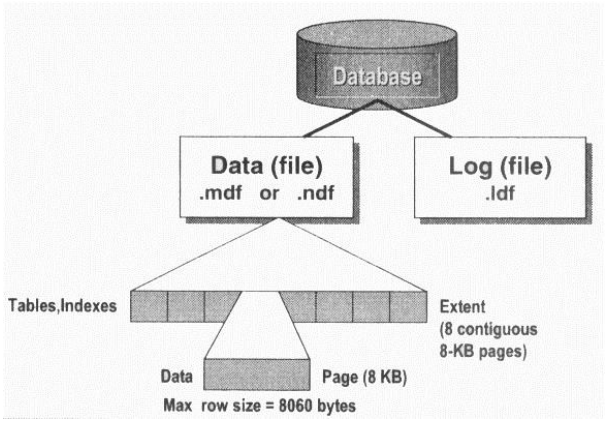
3 Database ontwerp - DDL

3.1 Database

3.1.1 Een database creëren

Bij creatie van een DB worden fysiek 2 files gemaakt: een datafile (mdf) en een logfile (ldf). Er kunnen meerdere datafiles zijn, nl. de secondary data files (ndf). Er kunnen ook meerdere logfiles zijn.

Bij creatie van een DB wordt een kopie genomen van de model database, deze bevat systeemtabellen en systeemviews.



Figuur 3: Opbouw van een database

Data wordt opgeslagen in blokken van 8 KB aaneensluitende diskspace, dit noemt met een pagina. Eén rij kan niet op meerdere pagina's bewaard worden, een rij mag maximaal

8060 b groot zijn (8KB – ruimte overhead). 8 opeenvolgende pagina's worden 1 extend (64 KB). Een tabel, index wordt opgeslagen in een extend. Dit alles is te zien op figuur 3

Logfiles bevatten informatie nodig voor recovery. De logfile-grootte is per default 25% van grootte van de datafile.

In sql creëert men een database als volgt:

```
CREATE DATABASE database_name
```

Men kan hierbij ook parameters meegeven:

```
CREATE DATABASE oefenDB
ON PRIMARY(
        NAME = oefenDB_data,
        FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL
            .1\MSSQL\Data\oefenDB.mdf',
        SIZE = 10MB, MAXSIZE = 15MB, FILEGROWTH = 20%)
LOG ON (
        NAME = tttoefenDB_log,
        FILENAME = 'C:\Program Files\Microsoft SQL Server\ MSSQL
            .1\MSSQL\Data\oefenDB.ldf',
        SIZE = 3MB, MAXSIZE = 5MB, FILEGROWTH = 1MB)
```

3.1.2 Een database verwijderen

```
DROP DATABASE database_name
```

Hierbij dient opgemerkt te worden dat de systeem databank niet verwijderd kan worden.

3.1.3 Een database wijzigen

- beheer van de groei van de database en log file
- uitbreiden/verminderen van grootte van database en log
- toevoegen/verwijderen van secondary database files, log files

Wijzigen van de groote van het logbestand:

```
ALTER DATABASE oefenDB
MODIFY FILE (name = 'oefenDB_log', size = 10MB)
```

Toevoegen van een databastand:

```
ALTER DATABASE oefenDB
ADD FILE (
        name = oefenDB2,
        filename = 'C:\Program Files\Microsoft SQL Server\ MSSQL
            .1\MSSQL\Data\oefenDB2.ndf',
        size = 10MB,
        maxsize = 15MB)
```

3.2 Tabellen

3.2.1 Een tabel creëren

Bij de creatie van een tabel specifieren we:

- de naam van de tabel.
- de definitie van de kolommen (naam, datatype ...).
- definitie van de constraints.

```
CREATE TABLE student(
        studentno int NOT NULL,
        lastname varchar(30) NOT NULL,
        firstname varchar(30) NOT NULL,
        gender char(1) NOT NULL,
        photograph image NULL)
```

3.2.2 Een tabel wijzigen

Mogelijke wijzigingen aan een tabel omvatten:

- toevoegen van kolommen.
- wijzigen van kolommen.
- verwijderen van kolommen.

```
ALTER TABLE student(
        ADD address varchar(40) NULL,
        ALTER COLUMN address varchar(50) NULL,
        DROP COLUMN address)
```

3.2.3 Een tabel verwijderen

Bij het verwijderen van een tabel diet men rekening te houden met de afhankelijkheden.

```
DROP TABLE student
```

3.3 Contraints

3.3.1 Identity waarden

Een identity kolom bevat voor elke rij een unieke waarde, die sequentieel door het systeem gegenereerd wordt. Er is slechts één identity kolom per tabel mogelijk. De identity kolom kan niet NULL zijn en kan niet door gebruikers aangepast worden.

```
CREATE TABLE studentVoorbeeldIdentity(
        studentno int identity(100, 5) NOT NULL,
        lastname varchar(30) NOT NULL,
        firstname varchar(30) NOT NULL,
        gender char(1) NOT NULL,
        photograph image NULL)
```

3.3.2 Data integriteit

Soorten:

- domein integriteit
- entity integriteit
- referentiële integriteit

3.3.3 Definitie van constraints

Via create table en als onderdeel van de kolomdefinitie:

```
CREATE TABLE studentVoorbeeldIdentity(
        studentno int NOT NULL unique)
```

Via alter table en als aparte lijn:

```
ALTER TABLE studentVoorbeeldIdentity(
        CONSTRAINT studentno_U unique(studentno)
```

Zowel bij creatie al bij wijzigen kan gekozen worden voor onderdeel van de kolomdefinitie als voor aparte lijn. NULL en DEFAULT constraints kunnen enkel bij definitie van de kolom worden opgegeven.

3.3.4 Check constraint

```
gender char(1) default 'M' check(gender in ('M','F')) not null
```

3.3.5 Primary key constraint

```
studentno int primary key
```

of

```
constraint studentno_PK primary key(studentno)
```

3.3.6 Foreign key constraint

De foreign key wordt gebruikt om verbanden tussen relaties uit te drukken. NULL waarden zijn niet toegelaten.

```
constraint class_fk foreign key(class) references class(classID)
```

De foreign key legt ook de trapsgewijze (cascading) referentiële integriteitsacties vast.

3.4 Views

3.4.1 Introductie

Een view is een SELECT statement die onder een eigen naam wordt bewaard. Een view is bijgevolg een soort virtuele tabel samengesteld uit andere tabellen of views.

De voordelen hiervan zijn dat de complexiteit van de database verborgen is. Gebruikers krijgen functionaliteit en rechten op maat. Views vereenvoudigen de beveiliging van de database. Data wordt ook georganiseerd voor de export naar andere applicaties.

```
CREATE VIEW view_name [(column_list)]
AS select_statement
```

Views kunnen net als tabellen verwijders en gewijzigd worden.

3.5 Indexen en performatie

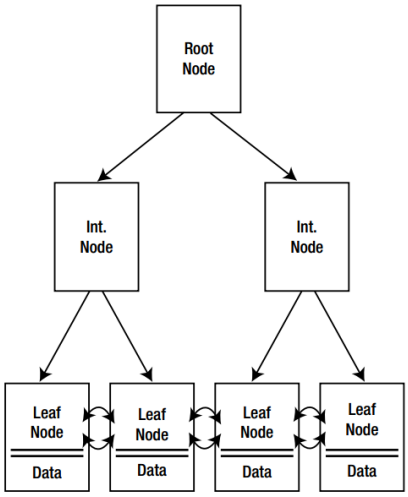
De heap is een ongeordende verzameling van data-pages zonder clustered index. Dit is de standaard opslag van een tabel.

Een index is een geordende structuur die op de records uit een tabel wordt gelegd. Deze zijn snel toegankelijk dankzij een boomstructuur. Dankzij indices kan de toegang tot data versnelt worden en kan uniciteit van de rijen afgedwongen worden. Daarentegen nemen indexen veel opslagruimte in beslag en ze kunnen de performantie ook doen dalen.

3.5.1 Clustered Index

De fysische volgorde van de rijen in een tabel is deze van de clustered index. Elke tabel kan maar één clustered index hebben.

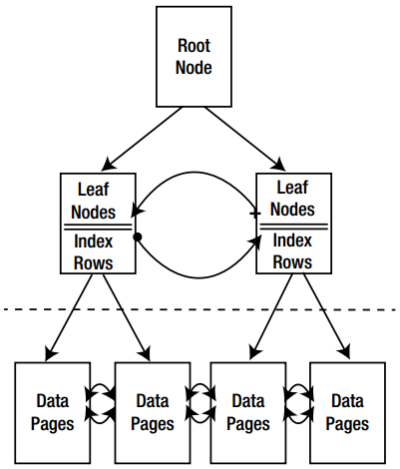
De voordelen t.o.v. table scan zijn dat een dubbel gelinkte lijst zorgt voor de volgorde bij het lezen van sequentiële records. Er zijn ook geen forward pointers. De clustered index legt unieke waarden op.



Figuur 4: Clustered index

3.5.2 Non-clustered Index

De non-clustered index is de default index, deze werkt trager dan de clustered index. Wel zijn er meerdere non clustered indexen mogelijk per tabel. Elke leaf bevat een sleutelwaarde en een row locator, deze wijst naar de positie in de clustered index als die bestaat, anders naar de heap.



Figuur 5: Non-clustered index

3.5.3 Covering index

Als een non-clustered indec een query niet volledig covert, dan moet de databank voor elke rij een lookup doen om de data op te halen. Een covering index is in wezen een non-clustered index die alle kolommen bevat die nodig zijn voor een bepaalde query.

3.5.4 Operaties op een index

Creatie:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
       INDEX index_naamON tabel(kolom[,...n])
```

Verwijderen:

```
DROP INDEX table_name.index_name[,...n]
```

4 Postrelationele DBMS

4.1 SQL als volwaardige taal

Procedurele uitbreidingen op SQL zijn merkgebonden. Database systemen die gebruik maken van deze uitbreidingen zijn dus moeilijker om te zetten van één RDBMS op een andere.

In het vervolg van deze samenvatting gaan we verder met t-SQL, die gebruikt wordt bij SQL Server.

De meeste database systemen voorzien al een aantal standaard SQl functies zoals minimum, maximum, som, gemiddelde, aantal ...

SQL Server heeft nog een aantal extra functies zoals datediff, substring, len, round ...

Naast deze functies is het ook nog mogelijk om user defined functies aan te maken.

4.1.1 Voordelen van PSM

- Code modularisatie
  - reduceren redundante code: veel gebruikte queries in Stored Procedures en hergebruiken in 3GL.
  - vaak voor de CRUD-operaties
  - minder onderhoud bij schema-wijzigingen
- Security
  - rechtstreekse queries op tabellen zijn uitgesloten
  - via stored procedures vastleggen wat kan en wat niet
  - vermijd SQL-injection, door gebruik input-parameters
- Centrale administratie van (delen van) DB-code.

4.1.2 Nadelen van PSM

- Beperkte schaalbaarheid
- Vendor lock-in
  - Er is geen standaard syntax
- Twee programmeertalen
- Twee debugomgevingen
- Beperkte OO-ondersteuning

4.1.3 Vuistregels

- Vermijd PSM voor grotere business logica
- Gebruik PSM voor technische zaken: logging, auditing, validatie
- Maak keuze voor portabiliteit/vendor lock-in in overleg met de business of corporate IT policies.

4.1.4 Stored procedure

Een stored procedure is een benoemde verzameling sql en control-of-flow opdrachten (programma) die opgeslagen wordt als een database object. De stored procedure is analoog aan procedures uit andere talen. Het kan aangeroepen worden vanuit een programma, trigger of stored procedure.

4.2 Variabelen

4.2.1 Lokale variabelen

```
DECLARE @lname varchar(40), @rijtelling int
SET @lname = 'Ringer'
SELECT @rijtelling = count(*)
FROM Authors
Where au_lname = @lname
PRINT 'Aantal werknemers met naam ' + @lname + ' = ' + str(
    @rijtelling)
```

Merk bij bovenstaande code op dat de naam van een variabele steeds wordt voorafgegaan door @. Het toekennen van een waarde gebeurt via SET of SELECT

4.2.2 Control flow met Transact SQL

In een programma kan je het verloop bepalen via o.a.:

- Instructie niveau
  - BEGIN ... END
  - IF ... ELSE
  - WHILE ...
    - BREAK
    - CONTINUE
  - RETURN
- Rij niveau
  - CASE ... END

4.2.3 Gebruik van commentaar

- Inline commentaar
  - –commentaar
- Block commentaar
  - /*commentaar*/

4.3 Cursors

SQL statements werken standaar met een complete resultaatset en niet met individuele rijen. Cursors laten toe om met individuele rijen te werken. Een cursor is dus een database object die wijst naar een resultaatset. Via de cursor kan men aangeven met wlke rij uit de resultaatset men wenst te werken.

4.3.1 Declaratie van een cursor

DECLARE <cursor_name> [**INSENSITIVE**] [**SCROLL**] CURSOR FOR
<SELECT_statement>
[FOR {**READ ONLY** | **UPDATE**[OF <**column** list >]}]

- INSENSITIVE
  - de cursor werkt met een tijdelijke kopie van de gegevens
    - wijzigingen in onderliggende tabellen worden niet gereflecteerd in gegevens opgehaald via de cursor.
    - de cursor kan niet gebruikt worden om tabellen te wijzigen

- wanneer INSENSITIVE weggelaten wordt dan worden deletes en updates wel degelijk gereflecteerd in de cursor.

  Dit is wel minder performant aangezien elke fetch nu resulteert in en nieuwe select opdracht.

- SCROLL
  - alle soorten fetch operaties zijn bruikbaar
    - FIRST, LAST, PRIOR, NEXT, RELATIVE en ABSOLUTE
  - wanneer SCROLL weggelaten wordt dan kan je enkel via NEXT data ophalen.
- READ ONLY
  - verhindert dat je via de cursor de onderliggende tabellen kan wijzigen
  - per default kan je via de cursor wel de onderliggende tabellen aanpassen
- UPDATE
  - benoemen van specifieke kolommen die kunnen gewijzigd worden via de cursor.

    Enkel kolommen benoemd in deze clause kunnen gewijzigd worden.

4.3.2 Openen van een cursor

```
OPEN <cursor name>
```

Hiermee wordt de cursor geopend en vervolgens opgevult met het SELECT statement dat in de declaratie was meegegeven.

De cursors current row pointer wordt gepositioneerd net voor de eerste rij in de actieve set.

4.3.3 Data ophalen via een cursor

```
FETCH [NEXT | PRIOR | FIRST | LAST | {ABSOLUTE|RELATIVE <
    rownumber>}]
FROM <cursor name>
[INTO <variable name> [,... <last variable name>]]
```

- De cursor wordt gepositioneerd
  - op de "volgende"(of vorige, eerste, laatste...) rij
  - per default wordt gepositioneerd via NEXT, voor andere manieren moet je een SCROLL-able cursor gebruiken.
- De gegevens worden opgehaald
  - zonder INTO clause worden resulterende gegevens op het scherm getoond.

- met INTO clause worden gegevens in de opgegeven variabelen gestopt. Hierbij moet men opletten dat de variabelen gedeclareerd zijn.

Een voorbeeld:

```
DECLARE @au_lname varchar(40), @au_fname varchar(20)
FETCH NEXT FROM authors_cursor
INTO @au_lname, @au_fname
WHILE @@FETCH_STATUS = 0 BEGIN
        PRINT 'Author: ' + @au_fname + ' ' + @au_lname
        FETCH NEXT FROM authors_cursor
        INTO @au_lname, @au_fname
END
```

4.3.4 Sluiten van een cursor

CLOSE <cursor_name>

- de cursor wordt gesloten
  - de definitie van de cursor blijft bestaan
    - er mogelijkheid om de cursor te heropenen

4.3.5 Dealloceren van een cursor

DEALLOCATE <cursor_name>

- de cursordefinitie wordt verwijderd
  - wanneer dit de laatste referentie naar de cursor was dan worden alle resources voor die cursor vrijgegeven
  - indien de cursor nog niet gesloten is da deallocate de cursor automatisch sluiten
    - een close opdracht net voor een deallocatie opdracht hoeft dus niet

4.3.6 Updaten via een cursor

```
UPDATE <table name>
SET ...
WHERE CURRENT OF <cursor name>
```

4.3.7 Verwijderen via een cursor

```
DELETE FROM <table name>
WHERE CURRENT OF <cursor name>
```

4.4 Stored Procedures

4.4.1 Creatie van een SP

```
CREATE PROCEDURE <proc_name> [parameter declaratie]
AS
<sql_statements>
```

- aanmaken db-object: via DDL instructie
- controle op syntax
  - enkel indien syntactisch correct wordt de stored procedure opgeslaan in de catalogus.

4.4.2 Wijzigen van een SP

```
ALTER PROCEDURE <proc_name> [parameter declaratie]
AS
<sql_statements>
```

4.4.3 Verwijderen van een SP

```
DROP PROCEDURE <proc_name>
```

4.4.4 Uitvoeren van een SP

```
EXCUTE <proc_name> [parameters]
```

- bij eerste uitvoering
  - compilatie en optimalisatie
- hercompilatie forceren
  - wenselijk bij wijzigingen aan structuur databank

```
execute uspOrdersSelectAll with recompile

execute sp_recompile uspOrdersSelectAll
```

4.4.5 De returnwaarde van een SP

- Bij uitvoering keert een SP een returnwaarde terug
  - deze waarde is een int
  - de default return waarde is 0.
- return statement

- uitvoering van de SP wordt gestopt
- laat toe om de returnwaarde te bepalen

Creatie van een SP met expliciete returnwaarde:

```
CREATE PROCEDURE usp_OrdersSelectAllAS
select * from orders
return @@ROWCOUNT
```

Gebruik van een SP met een returnwaarde:

```
DECLARE @returnCode int
EXEC @returnCode = usp_OrdersSelectAll
PRINT 'Er zijn ' + str(@returnCode) + ' records.'
```

4.4.6 SP met parameters

Via een input parameter kan men een waarde doorgeven aan de SP.

Via een output parameter kan men eventueel een waarde doorgeven aan de SP en krijgt men een waarde terug van de SP.

```
CREATE PROCEDURE usp_OrdersSelectAllForCustomer
@customerIDnchar(5) = 'ALFKI',
@count intOUTPUT
AS
SELECT @count = count(*)
FROM orders WHERE customerID= @customerID
```

Merk op dat 'ALFKI' een default waarde is die ingesteld wordt.

- aanroepen van de SP
  - voorzie steeds het keyword OUTPUT voor output parameters
  - twee manieren om actuele parameters door te geven
    - gebruik formele parameternaam
    - positioneel

Parameters via formele naam doorgeven:

```
DECLARE @aantal int
EXECUTE usp_OrdersSelectAllForCustomer
@customerID= 'ALFKI',
@count= @aantal OUTPUT
PRINT @aantal
```

Parameters positioneel doorgeven:

```
DECLARE @aantalint
EXEC usp_OrdersSelectAllForCustomer 'ALFKI', @aantalOUTPUT
PRINT @aantal
```

4.4.7 Error handling

@@erroris een systeemfunctie die het foutnummer bevat van de laatst uitgevoerde opdracht. De waarde 0 wijst op succesvolle uitvoering.

Alle foutboodschappen zitten in de systeemtabel sysmessages.

```
SELECT * FROM master.dbo.sysmessages
WHERE error = @@ERROR
```

Eigen fouten kan men genereren via raiseerror(msg,severity,state)

4.5 Triggers

Trigger is gekoppeld aan een database actie.

4.5.1 Procedurele database objecten

Soort	Batches	Opgeslaan als	Uitvoering	Ondersteunt parameters
Script	meerdere	Apart bestand	Client tool	nee
Stored Procedure	1	Database Object	Applicatie of SQL	ja
User defined function	1	Database Object	Applicatie of SQL	ja
Trigger	1	Database Object	DML, DDL, login-logoff statement	nee

Tabel 1: Procedurele programma's

Een batch is een programma zonder interactie.

4.5.2 Definitie

Een trigger is een speciaal type Stored Procedure.

Het is een stuk code dat automatisch uitgevoerd wordt als een neveneffect van een actie op een tabel.

4.5.3 Gebruik

- Validatie van data en complexe constraints
- Automatische generatie van waarden
- Ondersteuning voor alerts
- Bijhouden van audits
- Replicatie en gecontrolleerd bijhouden van redundante data

4.5.4 Syntax

```
CREATE TRIGGER TriggerName
BEFORE | AFTER | INSTEAD OF
INSERT | DELETE | UPDATE [OF TriggerColumnList]

ON TableName
[REFERENCING {OLD | NEW} AS {OldName | NewName}]
[FOR EACH {ROW | STATEMENT}]
[WHEN condition]
<trigger action>
```

Trigger Event:

- bepaalt bij welke gebeurtenis de trigger geactiveerd zal worden.
- bij update kan eventueel gespecifiëerd worden voor welke kolommen de triggering event gegenereerd wordt.

Trigger Condition

- wanneer de conditie false oplevert wordt er verder niets met de trigger gedaan.
- wanneer de conditie true oplevert zal de trigger action uitgevoerd worden.

Trigger Action:

- Een aantal SQL-opdrachten

Timing:

- **Before**: evaluatie van de trigger conditie en eventuele uitvoering van trigger actie gebeurt op de toestand van de DB zoals deze is vóór de triggering event zelf wordt afgehandeld.
- **After**: evaluatie van de trigger conditie en eventuele uitvoering van trigger actie gebeurt op de toestand van de DB zoals deze is nadat de triggering event zelf wordt afgehandeld.
- **Instead Of**: trigger wordt uitgevoerd ter vervanging van het DML statement.

Uitvoering:

- **For Each Row**: trigger wordt 'gewekt' voor elk tupel dat wordt gewijzigd door het trigger event.
- **For Each Statement**: 1 keer 'gewekt' ongeacht het aantal tupels dat gewijzigd wordt door het trigger event.

Verwijzen naar tupels/tabellen in de trigger

- **Old/oldName**: in de condition en trigger action kan verwezen worden naar de oude waarden van de tupels. Dit is niet relevant bij de insert trigger.
- **New/newName**: in de condition en trigger action kan verwezen worden naar de nieuwe waarden van de tupels. Dit is niet relevant bij delete trigger

4.5.5 Uitvoering van een trigger

Volgorde bij een true-condition:

1. uitvoering van beforestatement level triggersop de tabel
2. voor elke rij geaffecteerd door de trigger
   (a) uitvoering van beforerowlevel triggers
   (b) uitvoering van de triggering event(i.e. update/delete/insert)
   (c) toepassen van referentiële constraints
   (d) uitvoering van afterrowlevel triggers
3. uitvoering van afterstatement level triggersop de tabel

4.5.6 Voordelen

Mogelijkheid om functionaliteit in de DB op te slaan en consistent uit te voeren bij elke wijziging aan de DB.

Dus:

- geen redundante code: functionaliteit op 1 plaats
- wijzigingen aanbrengen wordt eenvoudig
- veiligheid: triggers in de database
- meer processing power
- past in client-server model

4.5.7 Nadelen

- Complexiteit: moeilijk om te debuggen en functionaliteit van applicatie naar DB

- Verborgen functionaliteit: onverwachte neveneffecten en cascading
- Performantie: Bij elke wijziging aan de DB moet de trigger conditie geëvalueerd worden.
- Portabiliteit: Dialect van DBMS

4.5.8 MS SQL Server: Trigger test tabellen

Er zijn 2 tijdelijke tabellen: de deleted en inserted tabel.

De deleted tabel bevat kopies van de gewijzigde of verwijderde rijen:

- tijdens de update of delete worden rijen van de trigger tabel gekopieerd naar de deleted tabel.
- deze twee tabellen hebben geen gemeenschappelijke rijen.

De inserted tabel bevat kopies van gewijzigde of ingevoegde rijen:

- tijdens een update of insert wordt een kopie van elke rij die gewijzigd of toegevoegd wordt in de trigger tabel geplaatst in de inserted tabel.
- deze twee tabellen hebben dus enkel gemeenschappelijke rijen.

Opmerkingen:

Naast verschillen in syntax, verschillen de SQL-producten tevens voor wat betreft de functionaliteit van triggers. Enkele interessante vragen hierbij:

- Mogen voor één tabel en een bepaalde mutatie meerdere triggers gedefinieerd worden?
- Kan de verwerking van een instructie die behoort tot een trigger actie leiden tot het activeren van een andere trigger?
- Wanneer wordt nu precies een trigger actie verwerkt?
- Mogen triggers gedefinieerd worden op catalogustabellen?

4.6 OODBMS en ORDBMS

4.6.1 Inleiding tot ORDBMS

De relationele DBMS is dominant, terwijl de OODBMS relatief klein is op de markt.

4.6.2 Enkele OO uitbreidingen

- gebruiker gedefinieerde types
- encapsulatie

- overerving
- polymorfisme
- dynamische method binding
- complexe objecten
- object identiteit

4.6.3 Realiteit

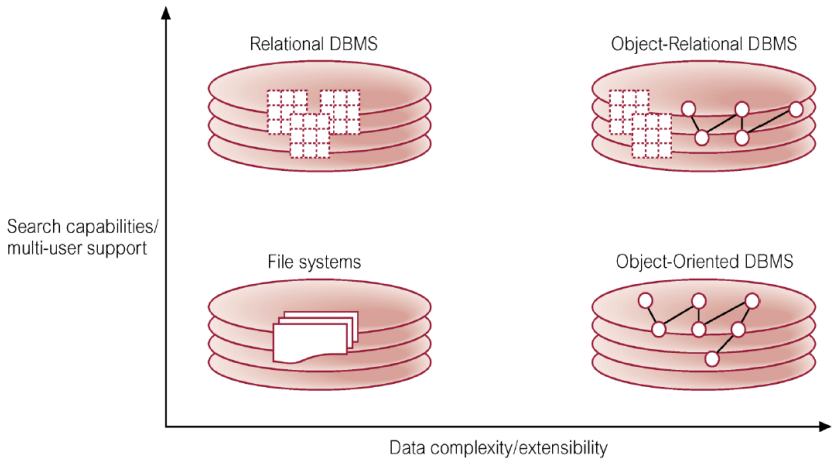
Er is geen standaard ëxtended"relationeel model.

Alle modellen

- hebben als basis relationele tabellen en query language
- hebben 1 of ander concept van object
- kunnen methodes opslaan

ORDBMS behouden de opgebouwde kennis rond RDBMS.

4.6.4 Stonebreakers View



Figuur 6: Stonebreakers View

4.6.5 Voordelen van ORDBMS

Oplossing voor de grote zwaktes van het relationeel model

- zwakke representatie van 'real-world' entiteiten
- semantische overloading
- zwakke ondersteuning voor integriteit en andere constraints

- homogene data structuur
- beperkte operaties
- geen recursie
- impedance mismatch
- hergebruiken delen
  - uitbreiden van de server functionaliteit om functionaliteit centraal uit te voeren
  - de centraal gedefinieerde functionaliteit kan nu gebruikt worden door alle applicaties
- met als gevolg een hogere productiviteit
- behoud van expertiseen kennisvan RDBMS
- backwardscompatibility

4.6.6 Nadelen van ORDBMS

- complexiteit van het OR model: eenvoud en puurheid van het relationeel model gaat verloren
- verhoogdekosten
- enkel een klein deel van de applicaties kan gebruik maken van de object extensies
- het is niet puur OO
  - OO applicaties zijn niet zo data gecentreerd
  - in pure OO zijn objects first class citizens, geen extensies van het relationele model
- SQL is te complex geworden

4.6.7 Impedance Mismatch

Er zijn verschillen tussen OO-model en relationeel model.

Dit probleem kan opgelost worden met OR-mapping of met User defined types.

4.7 User Defined Types

User defined types zijn abstracte types. Deze types kunnen gebruikt worden als built-in type.

Er zijn twee soorten:

1. distinct types

2. structured types

4.7.1 Distinct types

Een distinct type is gebaseerd op een basis type. Het laat toe om onderscheid aan te brengen tussen anders gelijke basis types.

4.7.2 Structured Types

Table Types

Table types worden opgeslagen in de DB. Deze types zijn tabellen.

Table variables bestaan slechts voor de duur van de batch.

- Voordelen:
  - kortere en overzichtelijke code
  - table type variabelen kunnen als parameters doorgegeven worden aan stored procedures en functions

Abstract Data Types

In het algemeen bevat een abstract data type:

- definitie van de attributen
- definitie van de routines (methodes)

Abstracte data types worden gebruikt om objecten op te slaan.

4.7.3 Overerving

Via UNDER kunnen subtype/supertype verbanden gedefinieerd worden.

Multiple inheritance is niet mogelijk.

Het subtype erft de attributen en methodes van het supertype. Het subtype kan uitgebreidt worden via definitie van nieuwe attributen en methodes.

Overloading van methodes is mogelijk.

Een supertype kan altijd door zijn subtype gesubstitueerd worden.

Men kan aan een ADT methodes toevoegen en deze implementeren in het CREATE TYPE BODY statement.

NOT FINAL: er kunnen nog subtypes gemaakt worden.

FINAL: er kunnen geen substypes gemaakt worden. (DEFAULT)

4.8 Large Objects

Een large object is een datatype die een grote hoeveelheid aan data kan vasthouden.

Er zijn verschillende soorten large objects:

- Binary Large OBject (BLOB)
- Character Large OBject (CLOB)
- National Character Large OBject (NCLOB)

De problemen met veel LOB's, die nu gebruikt worden in verschillende DBMS zijn:

- LOB's worden gezien als bytestreams
- DBMS kan zelf niets aanvangen met LOB's
- nadelig transfer van LOB'so p server naar client over het netwerk

5 Transactiebeheer

5.1 Inleiding

Een DBMS ondersteunt:

- transaction support
- concurrency control service
- recovery services

De bedoeling van transacties is dat de databank in een betrouwbare en consistentie toestand gehouden wordt.

- bij software/hardware failures
- bij gelijktijdig gebruik door meerdere gebruikers

5.2 Transacties

Een transactie is een actie, of een opeenvolging van acties op een DB die 1 logisch geheel vormen.

- Een actie: lezen of wijzigen van de inhoud van de DB. Lees operaties berokkenen anderen nooit problemen. Je kan echter wel last hebben van anderen.
- Een transactie wordt uitgevoerd door een gebruiker of door een programma.
- Het is een logische eenheid van werk

5.2.1 Resultaat van een transactie

1. commited
    - De transactie kent een succesvolle afloop
    - De transactie commit en de DB heeft een consistente toestand bereikt.
2. aborted
    - De transactie is niet succesvol afgehandeld
    - de transactie abort en de DB moet teruggebracht worden naar de consistente toestand waarin ze zich bevond voor de transactie werd gestart.

      Dit gebeurt via ROLLBACK of UNDO

Een committed transaction kan nooit geaborted worden.

Een aborted transaction kan herstart worden.

5.2.2 ACID Eigenschappen

Atomicity: de opdrachten van een transactie worden als één ondeelbaar geheel beschouwd.

- alles of niets
- DBMS verantwoordelijkheid: recovery subsystem

Consistency: een transactie brengt de DB van de een consistente toestand naar een andere consistente toestand.

- DBMS verantwoordelijkheid énprogramma verantwoordelijkheid

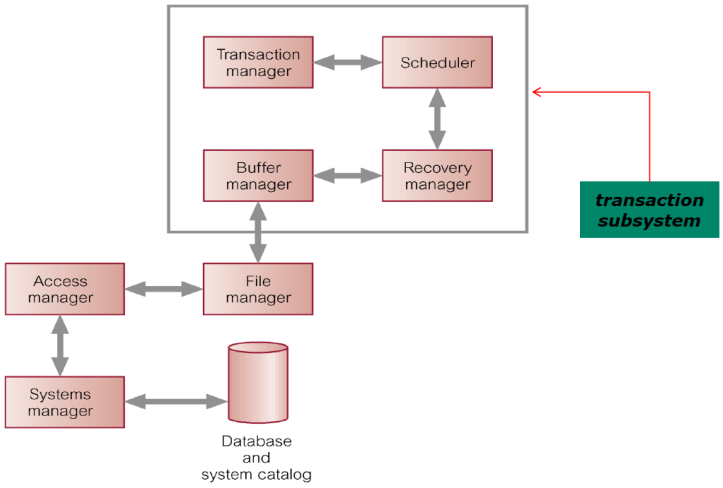
Isolation: transacties worden onafhankelijk van elkaar uitgevoerd.

- transacties mogen niet ongewenst interfereren met elkaar
- transacties mogen geen uitkomsten aan andere transacties presenteren vóór de commit
- DBMS verantwoordelijkheid: concurrency-controlsubsystem

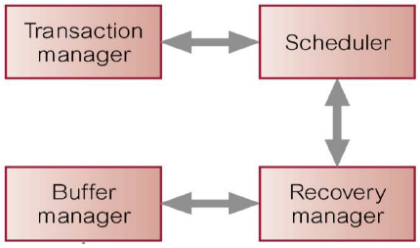
Durability: na een COMMIT zijn de aangebrachte wijzigingen gegarandeerd permanent

- na een storing kunnen deze wijzigingen steeds worden gerecupereerd
- DBMS verantwoordelijkheid: recovery subsystem

5.2.3 Databank architectuur - Transaction support



Figuur 7: Databank architectuur - Transaction support



Figuur 8: Transactie subsysteem

- transaction manager
  - coördineert transacties van programma's
- scheduler
  - implementeert strategie voor concurrency control
  - akalock manager bij een op locking gebaseerde strategie
  - doel: concurrency maximaliseren zonder interferentie tussen verschillende transacties
- recovery manager
  - bij failures brengt deze de db terug in consistente toestand (vóór de transactie gestart werd)
- buffer manager
  - verantwoordelijk voor efficiënte transfer van data tussen main memory en disk storage

5.3 Concurrency Control

concurrency control is het beheer van gelijktijdige acties op de DB zonder dat ze interfereren met elkaar.

5.3.1 Waarom?

Concurrency Control voorkotm interferentie wanneer twee of meerdere transacties de databank gelijktijdig aanspreken, en minstens 1 transactie data wijzigt.

Hoewel twee transacties op zichzelf beschouwd correct kunnen zijn, kan het verweven van hun acties leiden tot een incorrect resultaat.

5.3.2 Problemen bij concurrency control

Lost update

Het lost update probleem treedt op wanneereen schijnbaar compleet succesvolle update van een transactie wordt overschreven door een andere transactie.

Uncommitted Dependency

Het uncommitted dependency (of dirty read) probleem treedt op wanneer een transactie de intermediaire resultaten van een andere, uncommitted transactie, kan zien.

Uncommited Dependency staat ook bekend als Dirty Read.

Inconsistent Analysis

Het inconsistent analysis probleem treedt op wanneer een transactie verschillende waarden uit de DB leest, terwijl een andere transactie bezig is sommige van deze waarden te veranderen.

Twee gerelateerde problemen aan de inconsistent analysis zijn:

1. non-repeatable read of fuzzy read
    - Twee keer lezen van eenzelfde item levert twee verschillende waarden op.
2. phantom read
    - Een transactie leest een aantal records die voldoen aan een bepaalde voorwaarde.
    - De transactie herhaalt dit later maar merkt nu dat er ondertussen andere records zijn bijgekomen via een andere transactie.

5.3.3 Serializability en recovery

Schedule

een schedule is een sequentie van acties van een aantal concurrente transacties die de volgorde van de acties van de individuele transacties respecteert.

Recoverability

De databank zelf schedules laten opstellen die zonder interferentie problemen concurrent kunnen worden uitgevoerd.

- ze garanderen de consistencyen isolationeigenschap van transacties
- dit is in de veronderstelling dat geen enkele van de transacties in de schedule faalt

recoverability:

- wanneer een transactie faalt moeten we de effecten ongedaan kunnen maken.
  - atomiciteitvan transacties
- wanneer een transactie commit zijn de veranderingen permanent
  - durability eigenschap van transacties

5.3.4 Locking

Locking is een methode gebruikt om concurrente toegang tot data te beheren.

Wanneer een transactie toegang heeft tot de DB kan er via een lock voor gezorgd worden dat andere transacties toegang tot de DB geweigerd worden om zo incorrecte resultaten te voorkomen.

Locking is de meest gebruikte manier om de consistentie te garanderen.

Basisregels voor locking

Een transactie kan een data item lezen of schrijven enkel en alleen als het een lock heeft verworven voor dat data element, en het bovendien deze lock nog niet heeft vrijgegeven.

Als een transactie een lock op een data item verwerft, dan moet het later ook die lock terug vrijgeven.

Soorten locks

Shared locks (S-Locks):

- een transactie met een shared lock op een data item kan dat data item lezenmaar niet wijzigen
- meerdere transacties kunnen gelijktijdig een shared lock op eenzelfde data item bezitten

Exclusive locks (X-Locks):

- een transactie met een exclusive lock op een data item kan dat data item lezen en wijzigen

- op elk ogenblik kan hoogstens 1 transactie een exclusive lock op een data item hebben

5.3.5 Deadlock

Een impasse die kan ontstaan wanneer twee of meerdere transacties elk wachten op het vrijgeven van locks die de andere transactie heeft.

Het oplossen van een deadlock gaat meestal als volgt: abort en restart 1 of meerdere transacties.

Timeouts

Een transactie die een lock aanvraagt wacht voor een bepaalde vooraf gedefinieerde tijdop die lock.

Indien lock niet toegekend tijdens dit interval

- Assumptie dat er een deadlock is.
  - hoewel dit niet noodzakelijk zo is...
- Aborten restartvan de transactie.

Deadlock Prevention

- Gebruik makend van transaction time-stamps
  - time-stampvoor deadlock detection
- T wacht op locks die U vast heeft
  - wait-diealgoritme
    - als T ouder is dan U dan zal T wachten
    - in ander geval 'sterft' T
      - abort/restart met dezelfde timestamp
  - wound-waitalgoritme
    - als T ouder is dan U dan zal het U 'verwonden'
      - meestal betekent dit een abort/restart van U
    - in andere geval zal T wachten

Deadlock Detection en Recovery

Detection:

- Gebruik makend van een wait-for graph met transactie afhankelijkheden
- Wanneer de wait-for graph een lus bevat is er een deadlock
- Op regelmatige tijdstippen wordt deze graph getest op lussen

Recovery:

- 1 of meerdere transacties worden ge-abort
- Problemen:
  - Victim Selection
    - abort die transactie waarvoor de abort een 'minimale kost' met zich meebrengt.
    - enkele parameters die kunnen gebruikt worden:
      - tijd dat de transactie al aan het runnen was
      - aantal data items dat reeds werd gewijzigd door de transactie
      - aantal data items die nog moeten worden gewijzigd door de transactie
  - hoe ver moet een transactie een rollback doen
    - dit is niet noodzakelijk de volledige transactie
  - voorkomen van starvation
    - komt voor wanneer steeds dezelfde transactie als victim wordt geselecteerd.
      - teller bijhouden

5.4 Transacties in SQL-server

Impliciete transacties:

- insert, update, delete
- elke transact sql-opdracht

Expliciete transacties:

- zelf aangeven waar transactie begint, wanneer de transactie kan afgesloten worden, of hoe je op je stappen terugkeert om fouten op te vangen.
  - BEGIN TRANSACTION
  - COMMIT TRANSACTION
  - ROLLBACK TRANSACTION

5.4.1 Isolation levels

Bepalen het gedrag van concurrent users die data lezen of schrijven.

- Reader: statement dat data leest, m.b.v. een shared lock
- Writer: statement dat data schrijft, m.b.v. een exclusive lock

Writers kun je niet beïnvloeden voor wat betreft de locks die ze nemen en de duur van de locks.

Readers kun je wel expliciet beïnvloeden, hierdoor heb je impliciet invloed op het gedrag van de writers.

Isolation level = setting op sessie-niveau of query-niveau.

5.4.2 Isolation levels in SQL-server

1. read uncommitted
2. read committed (default)
3. repeatable read
4. serializable

Hoe hoger het nummer hoe minder beperkingen, hoe lager het nummer hoe meer afscherming en dus meer hinder voor andere sql statements.

Read uncommitted

- laagste isolatieniveau
- reader vraagt geen shared lock
- reader nooit in conflict met writer, die exclusieve lock heeft
- reader lees incommited data (=dirty read)

Read committed

- default isolation level
- laagste niveau dat dirty reads verhindert
- reader leest enkel committed data
- reader vraagt hiervoor een shared lock
- als bij deze vraag een writer een exclusive lock heeft, moet de reader wachten op shared lock
- reader houdt shared lock tot de data verkregen is, niet tot einde van zijn transactie
  - nogmaals lezen van de tata in dezelfde transactie kan een ander resultaat opleveren

    = non-repeatable reads of inconsistent analysis
  - acceptabel voor veel toepassingen

Repeatable read

- reader vraagt shared lock en houdt deze tot het einde van de transactie.

- andere transactie kan geen exclusive lock verkrijgen tot einde van de transactie van de reader
- repeatable read = consistent analysis
- vermijd ook *lost update* door bij het begin van de transactie shared lock te nemen

Serializable

- repeatable read lockt enkel rijen gevonden door de eerste SELECT.
- zelfde SELECT in dezelfde transactie kan nieuwe rijen geven = *phantoms*
- Serializable vermijdt deze phantoms
- lockt alle keys die beantwoorden aan de WHERE-clause, ook toekomstige

5.4.3 Isolation levels: Sessie niveau

```
SET TRANSACTION ISOLATION LEVEL [isolationlevel]
```

5.4.4 Isolation levels: Query niveau

```
SELECT * FROM ORDERS WITH (NOLOCK);
of
SELECT * FROM ORDERS WITH (READUNCOMMITTED);
```

Dit vermijdt dat langlopende ad-hoc query's op productie-systeem updates in andere transacties laten wachten bij READ COMMITTED en hoger.

5.5 Recovery

Recovery is het proces waarbij een DB wordt teruggebracht naar een correcte toestand wanneer er zich een failure voordoet.

5.5.1 Media

- Main memory
  - random acces
  - online
  - vluchtig
  - onbetrouwbaar
  - duur
  - zeer snel
- Magnetic disc

  - random acces
  - online
  - niet vluchtig
  - onbetrouwbaar
  - minder duur
  - snel
- Tape (casette): voornamelijk backup's en zaken die weinig gebruikt worden
  - sequentieel
  - online of offline
  - niet vluchtig
  - zeer betrouwbaar
  - goedkoop
  - traag
- Optical disc
  - random acces
  - online of offline
  - niet vluchtig
  - zeer betrouwbaar
  - zeer goedkoop
  - traag

Stabiele opslag: replicatie op verschillende plaatsen

Mirroring: quasi tegelijk naar meerdere locaties gegevens wegschrijven

Virtuele tapedrives: gesimuleerde tapedrives op een disk en pas als er op de de virtuele tapedrive geen plaats meer is op de virtuele tape naar een reële tape wegschrijven. Dit zorgt voor perfomatie winst, omdat de write acties korter zijn naar de disk en omdat de tapes altijd volledig vol zijn. Er zijn dus nooit tapes die niet volledig gevuld zijn met data.

5.5.2 Soorten failures

System crash

- hardware of software errors
- verlies van gegevens in main memory

Media failure

- vb. disk head crasht
- verlies van gegevens in secondary storage

Software error in applicaties

- vb. logische fout die transacties doet falen

Natuurlijke rampen

- vb. brand, aardbeving

Slordigheid

- vb. onopzettelijk wissen van gegevens door de gebruiker of DB admin

Sabotage

- vb. opzettelijk wissen of corrupteren van gegevens of infrastructuur

5.5.3 Transactions en recovery

De eenheid voor recovery is een transactie.

De recovery manager staat voor:

- atomiciteit (**A**CID)
- duurzaamheid (ACI**D**)

De moeilijkheid hierbij is dat het schrijven naar een databank niet atomaire is. Een transactie kan committen zonder dat alle effecten al (permanent) in de DB geregistreerd zijn.

5.5.4 Undo en Redo

Enkel bij een flush van de buffer is de data permanent.

- **Flushing**: data overhevelen van primary storage naar disk storage

Expliciete flush

- force writing
- dit gebeurt via een commando, vb. commit

Impleciete flush

- wanneer de buffers vol zijn

Als er een failure is tussen het schrijven naar de buffer en de flushing dan:

- Transactie was reeds gecommit

  - durability: redo de wijzigingen
  - aka roll forward
- Transactie was nog niet gecommit
  - atomicity: undo de wijzigingen
    * partiële undo: undo van 1 transactie
    * globale undo: undo van alle actieve transacties
  - aka rollback

5.5.5 Buffer management

buffer management omvat het beheer van transfer van buffers tussen main memory en disk.

Praktisch:

- inlezen tot buffer vol
- replacement strategie voor force-write
  - FIFO: first in first out
  - LRU: least recently used

5.5.6 Recovery faciliteiten

Het DBMS beidt de volgende diensten aan:

- back-up mechanisme
  - periodische back-ups van de DB, dit gebeurt automatisch zonder dat de systemen moeten stoppen en de kopieën worden bewaard op offline storage.
  - compleet of incrementeel
- logging mogelijkheden
  - op de hoogte blijven van de huidige toestand van transacties en db wijzigingen
  - de log bevat mogelijks: transaction id, type of log record, id van de gewijzigde data, before image, after-image
  - log wordt ook gebruikt voor performance monitoring en auditing
  - log is belangrijk wordt in 2 of 3 voud bijgehouden, ook offline
  - log moet snel toegankelijk zijn: minor failures moeten direct opgelost worden $\rightarrow$ liefst ook op online storage
- checkpoint mogelijkheden

  - om lopende wijzigingen in de db permanent te maken
- recovery manager
  - om de db in een consistente toestand te brengen na een failure

5.5.7 Checkpointing

Een checkpoint is een synchronisatiepunt tussen de databank en de log, op dit punt worden alle buffers ge-flushed.

Checkpoints worden voorzien op vooraf ingestelde intervallen.

Een checkpoint omvat:

- alle log records in main memory wegschrijven naar disk
- de gewijzigde delen van de buffers wegschrijven naar disk
- een checkpoint record in de log registreren
  - dit record bevat id van alle transacties die actief zijn op het moment van checkpointing

5.5.8 Recovery technieken

Soort recovery procedure die gevolgd wordt hangt af van de ernst van het probleem.

- serieuze problemen
  - backup restoren
  - wijzigingen van committed transacties (sedert de backup) die verloren gingen, herstellen adhvde log
- problemen van inconsistentie
  - kan zonder backup
  - undo/redo adhvde log'sbeforeen afterimages

5.5.9 Deferred update

Bij een deferred recovery protocol worden wijzigingen van een transactie niet weggeschreven naar de dbzolang de transactie niet het commit-punt bereikt.

1. start van de transactie
   - registreert transaction start in de log
2. bij een write actie
   - registreert dit in de log: enkel after-image, geen before-image nodig

- registreert dit niet in de DB

3. commit van de transactie
    - registreert transaction commit in de log
    - schrijf alle log records weg naar disk (flush de log vóór de volgende stap!)
    - commit, update de DB adhv de log records
4. abort van de transactie
    - negeer de log records, schrijf niets naar disk

5.5.10 Immediate update

bij een immediateupdate recovery protocol worden wijzigingen van een transactie direct weggeschreven naar de DB.

1. start van de transactie
    - registreer transaction start record in log
2. bij een write actie
    - schrijf het log-record naar disk: het log-recordbevat before-image en after-image
    - bij flush van de buffers wordt de wijziging naar de DB geschreven
3. commit van de transactie
    - schrijf een transaction commit log record naar disk
4. abort van de transactie
    - gebruik before-images voor undo