Graphical User Interface

Begrippen overerving, inner classes (anonymous) en interface/implementatie

Overerving

- klassen met gemeenschappelijke eigenschappen, vb Docent, Student → Persoon
- uitbreiding van bestaande klasse, vb MijnJFrame → JFrame

Interface

- abstacte definitie van een type
- soort contract: klassen die interface implementeren:
  - zullen doen wat in contract staat
  - hoe ~ klasse die interface implementeert
- beschrijving van een aantal methoden (wat, niet hoe) die de implementerende klassen MOETEN hebben
- geen implementatie, geen instanties
- 1 klasse kan meerdere interfaces implementeren
- zoals class maar
  - enkel publieke instantiemethoden zonder implementatie (enkel hoofding)
- eventueel constanten (geen variabelen)

Geneste klassen en interfaces

- klassen en interface gedeclareerd binnen andere
- genest type is deel van omvattend type = vertrouwens(trust)relatie, elk kan alle members van andere bereiken
- reden:
    - type structureren
    - logisch gerelateerde objecten eenvoudig te connecteren
- eigenlijk zoveel mogelijk proberen te vermijden (niet te hergebruiken)
- zoals statische methoden, geen directe toegang tot instantievariabelen en –methoden, enkel via objectreferentie
- static nested class interageert met instantieleden van zijn outer class zoals alle andere top-level klassen
- static nested class gedraagt zich als top-level class dat in andere class genest is voor betere groepering
- nested interfaces altijd static (mag static weglaten)
- nested class in interface altijd static (mag static weglaten)

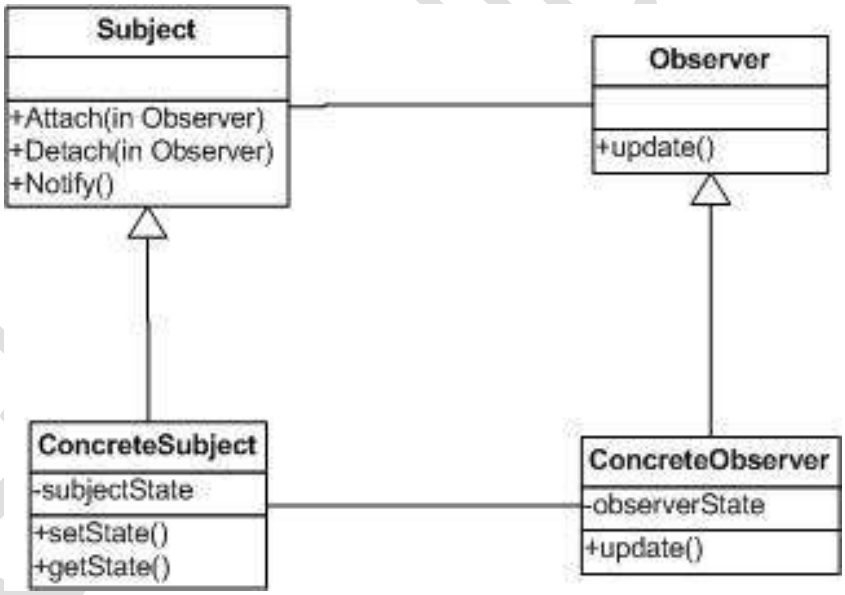
- non-static nested classes = innerclasses
    - geassocieerd met instanties: object van innerclass is altijd geassocieerd met object van omsluitende class
    - local inner class:
        - gedefinieerd in methode
        - onbereikbaar van buiten methode
    - anonymous inner class
        - op ogenblik van instantiatie ook definitie (1 instantie) met new supertype()
        - extends class of implements interface
        - geen expliciete constructoren

Observer patroon

Het observerpatroon voldoet aan deze doelstellingen door het instellen van de rollen Subject en Observer binnen het programma. Een Subject is een object waarvan de toestand gevolgd wordt door een Observer.

Het patroon realiseert minimale koppeling tussen objecten door de rolverdeling om te keren: een Observer registreert de toestandsverandering van het Subject niet; het Subject publiceert toestandsveranderingen aan geïnteresseerde Observers. Een Subject beschikt daartoe over een notify() methode die aan alle geïnteresseerde Observers het update()-bericht stuurt. Dit heeft als voordeel dat de Observers alleen een interface hoeven te hebben om veranderingen gepubliceerd te krijgen en verder niets over een Subject hoeven te weten - geen naam, geen informatie over de interne toestand, niets. Een Observer wordt zo een generiek iets dat alleen zijn eigen toestand moet kennen om een update uit te voeren, maar verder wellicht zelfs op verschillende typen Subject kan reageren.

Naast notify() beschikt Subject ook over methoden om geïnteresseerde Observers te verbinden aan het Subject of er weer van los te maken (attach() en detach()). Een geïnteresseerde Observer is verbonden en wordt door het Subject op de hoogte gesteld van veranderingen. Merk op dat ook Subject niet sterk gekoppeld is aan een ander object; een Subject hoeft alleen de naam (en het type) Observer te kennen.



Observer-interface: 1 methode: update

- public void update(Observable obj,Object arg)
- obj: dat wijzigt
- arg: mogelijkheid om iets door te geven van Observable naar Observer

Model-View-Controller patroon

Model-view-controller (of MVC) is een ontwerppatroon ("design pattern") dat het ontwerp van complexe toepassingen opdeelt in drie eenheden met verschillende verantwoordelijkheden: datamodel (model), datapresentatie (view) en applicatielogica (controller). Het scheiden van deze verantwoordelijkheden bevordert de leesbaarheid en herbruikbaarheid van code. Het maakt ook dat bijvoorbeeld veranderingen in de gebruikersinterface niet direct invloed hebben op het datamodel en vice versa.

Model

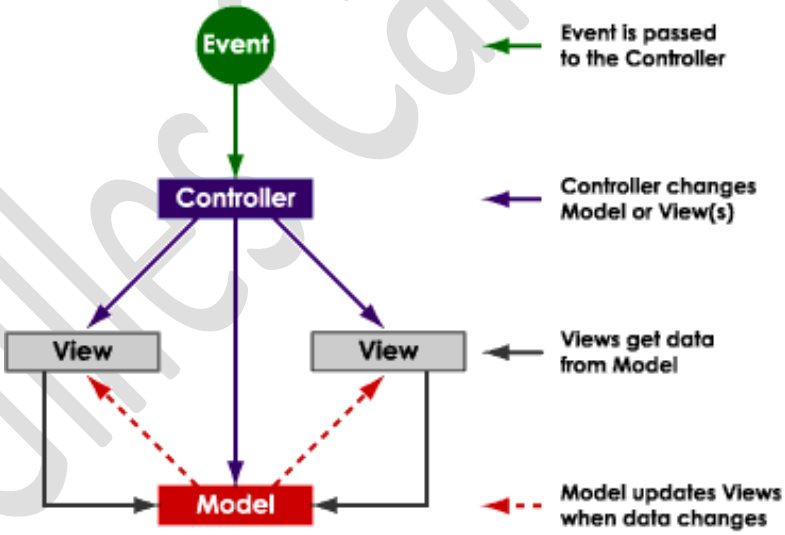
Definieert de representatie van de informatie waarmee de applicatie werkt. Aan ruwe gegevens wordt betekenis gegeven door relaties tussen data en logica toe te voegen. De daadwerkelijke opslag van data wordt gedaan met behulp van een persistent opslagmedium, zoals een database. De applicatie zal gegevens die gebruikt worden in het model, ophalen en wegschrijven van en naar de dataopslag via een datalaag. De datalaag is niet per se een onderdeel van het MVC-patroon.

View

Informatie wordt weergegeven via de View. Userinterface-elementen zullen gedefinieerd zijn in dit onderdeel. De view doet geen verwerking (zoals berekeningen, controles,...) van de gegevens die getoond worden.

Controller

De controller verwerkt en reageert op events, die meestal het gevolg zijn van handelingen van de gebruiker.



De controller kent de view en het model.

De view kent het model.

Het model heeft geen weet van eventuele views, wat het uitbreiden van de applicatie makkelijker maakt

Compileren en decompileren
Debuggen, profiling en testen

Compileren

Het omzetten van programmacode in tekstformaat naar formaat dat de computer –eventueel virtueel- verstaat.

Eerst worden de bestanden één voor één gecompileerd om nadien vaak nog via een link of build de delen aan elkaar te hangen.
Dit bij bijvoorbeeld C-programma's, deze worden gecompileerd naar .O bestanden om nadien te linken naar een .exe bestand.

Interpreteren

Een interpreter is een computerprogramma dat broncode van computerprogramma's vertaalt in een voor de processor begrijpelijke vorm en METEEN uitvoert – in tegenstelling tot een compiler-, die een programma's opslaat in een dergelijke vorm.

Bij herhaald gebruik van een programma is het ene nadeel dat een interpreter de instructies telkens opnieuw omzet, terwijl dat bij een compiler maar eenmaal gebeurt, en het programma in vertaalde vorm bewaard wordt voor een herhaalde uitvoering. Interpreteren is dan weer voordelig tijdens het testen van de code.

Decompileren

Dit is het ongedaan maken van het gecompileerde bestand.
Bijvoorbeeld bij Java wordt een .class bestand gecompileerd tot een .java bestand.

Obfuscators

Natuurlijk wil men vermijden dat men via decompilers de oorspronkelijke code terug kan achterhalen. Hierbij kan men gebruik maken van een obfuscator, zo wordt de gecompileerde code val gezet met 'junk' code die de algemene code slordig moet maken.

Java naar native code

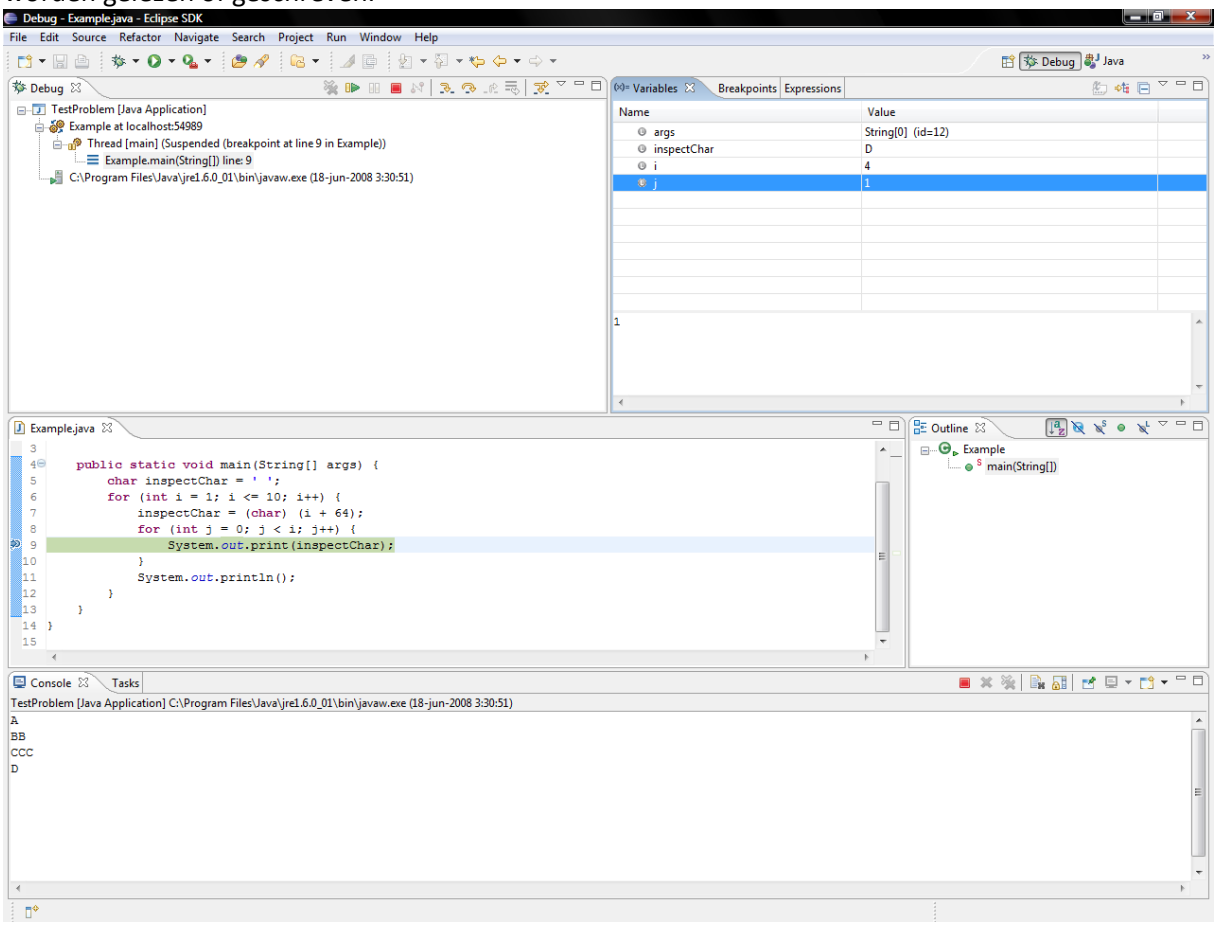
Het is ook mogelijk om Java te compileren naar een .exe file (Windows) en dit via bijvoorbeeld, Excelsior JET.

Debuggen

Debuggen is het opsporen en verhelpen van bugs in computerprogramma's en elektronische hardware.
Dit gebeurt door het stap-voor-stap doorlopen van de code.
Zo kan men breekpunten plaatsen om de code nader te bekijken.

Bij softwareontwikkeling is een breakpoint het opzettelijk stopzetten van de uitvoering van een computerprogramma om het te kunnen debuggen. De programmeur kan vervolgens de toestand van het programma op dat moment inspecteren (zoals de inhoud van het geheugen of van bestanden) om te kijken of het programma naar behoren functioneert.

Een breekpunt kan op een bepaalde plek in de broncode gezet worden zodat het programma gepauzeerd wordt wanneer de uitvoering bij die regel komt. Het is ook mogelijk een breekpunt te laten afhangen van enkele condities om het programma te stoppen als aan die condities is voldaan. Daarnaast kan men een breekpunt instellen als bijvoorbeeld bepaalde locaties in het geheugen worden gelezen of geschreven.



'Watching a variable' is het proces van het inspecteren van een waarde van een variabele en dat tijdens dat het programma aan het lopen is in debug mode.

Profiling

Profiling is het analyseren van het gedrag –behavior- van een programma tijdens de uitvoering.

Het doel hierbij is, het constateren van bepaalde delen van code waarbij het optimaliseren naar snelheid of geheugenverbruik nodig of nuttig zou zijn.

Men kan onder andere meten hoelang bepaalde delen van code uitgevoerd worden ne hoeveel keer een bepaalde methode is opgeroepen.

Testen (unit testen)

Men test een stuk code apart, bij OO, test men typisch elke klasse part, dus in elke klas is er een main die de klasse test.

Logging

Hierbij schrijft men extra informatie – naar het scherm, maar meestal naar ene logfile – voor vooral latere debugging of loggen van gebruikers en hun acties.

Men maakt gebruik van info op verschillende niveaus. Zo zal men in het begin van het programma veel info bij houden, maar naarmate het programma een bepaalde duur goed loopt, worden er alsmaar minder info opgeslagen. Zo is de info in begin vanboven weergegeven en de later info nadien onderaan in het log-niveau.

Alle verschillende klassen loggen ook altijd naar 1 bestand.

I/O

IO tussen programma's

Sockets

In software wordt de term socket gebruikt voor een standaard methode waarmee een programma met een ander computerproces communiceert.

Een socket is een voorbeeld van een application programming interface, kortweg API genoemd.

Een application programming interface (API) is een verzameling definities op basis waarvan een computerprogramma kan communiceren met een ander programma of onderdeel (meestal in de vorm van bibliotheken). Vaak vormen API's de scheiding tussen verschillende lagen van abstractie, zodat applicaties op een hoog niveau van abstractie kunnen werken en het minder abstracte werk uitbesteden aan andere programma's. Hierdoor hoeft bijvoorbeeld een tekenprogramma niet te weten hoe het de printer moet aansturen, maar roept het daarvoor een gespecialiseerd stuk software aan in een bibliotheek, via een afdruk-API.

De socket wordt geïdentificeerd door de combinatie van de poortnummer en het IP-adres, gescheiden door een deelteken (bijvoorbeeld 127.0.0.1:80). Deze unieke combinatie wordt ook wel het socketadres genoemd.

Poortnummer

Elk programma dat wilt communiceren over het netwerk maakt een socket aan. Aan de serverzijde worden hiervoor voor gedefinieerde poortnummers gebruikt. De client kiest zelf een willekeurige poortnummer (boven 1023) die nog niet in gebruik is.

Werkwijze

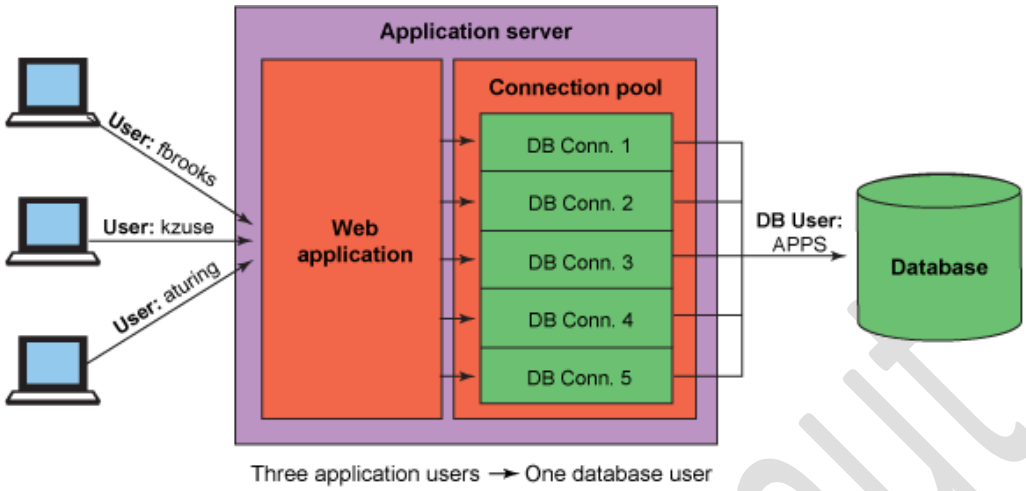
1. server maakt communicatie-eindpunt (=socket) via socket
2. server plakt gewenst poortnummer op socket via bind
3. server wacht op binnenkomende verzoeken (listen) via accept
4. client maakt communicatie-eindpunt (=socket) via socket, TCP-software OS kiest vrij poortnummer voor server-antwoord
5. client maakt contact met server via connect
6. dit schudt accept van server wakker
7. doorsturen gegevens via stromen op socketverbinding
8. connectie sluiten met close

Java

- import java.net.*
- class ServerSocket, constructor met poortnummer: socket + bind
- class Socket, constructor met machinenaam of -nummer en poortnummer server: socket + connect

Toegang tot databases vanuit programma's

Werken met een database connectie.



ODBC

Doel:
Toegang mogelijk maken tot gelijk welke data, van gelijk welke toepassing, dus onafhankelijk van gebruikte DBMS.

Hoe:
Via een laag tussen applicatie en DBMS, een database driver, vertaalt de dataqueries van de toepassing naar commando's die de DBMS verstaat.

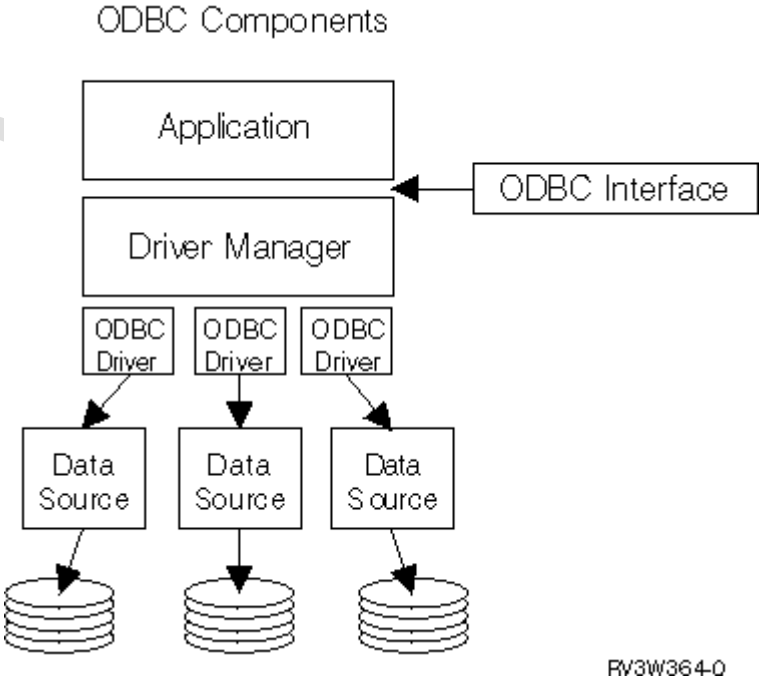
Laat programma's toe om SQL requests te gebruiken, zonder proprietary interfaces naar de database te kennen

ODBC converteert de SQL requests naar requests die het individuele database systeem verstaat.

Voorwaarde:
app. En DBMS moet ODBC-compliant zijn.

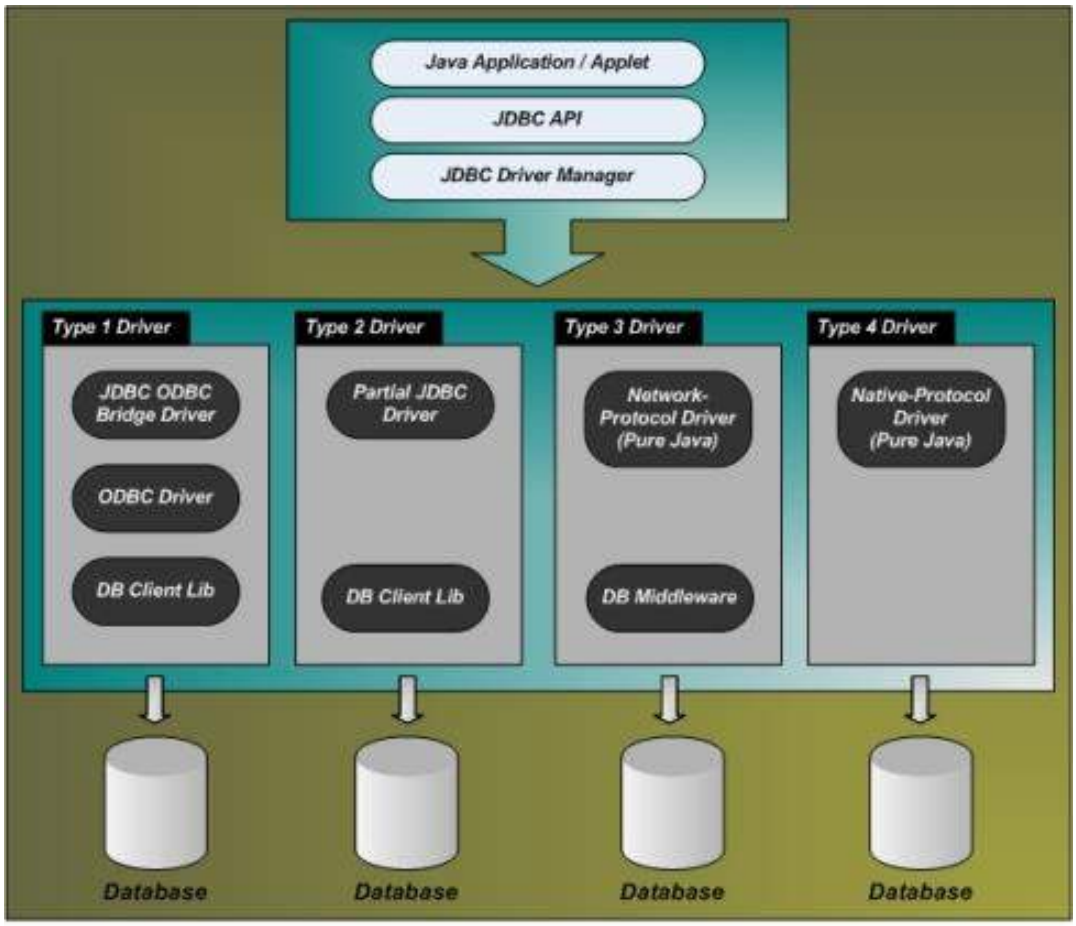
Native interface die nativa library oproept (DLL voor windows)

Nadeel:
Databank moet geregistreerd zijn (windows-niveau)



Java en JDBC[1]

Databank toegang vanuit Java



1. JDBC-ODBC bridge
2. Java en native driver
3. Java en middelware
4. Puur java

Voordelen:

- Eenvoudig
- Zelfs over netwerken (enkel URL nodig)
- Toegang tot metadata

[1] http://www.ustudy.in/node/5475

1. JDBC-ODBC

JDBC die ODBC-driver gebruikt voor connectie met databank

Driver vertaalt JDBC-oproepen naar ODBC-oproepen

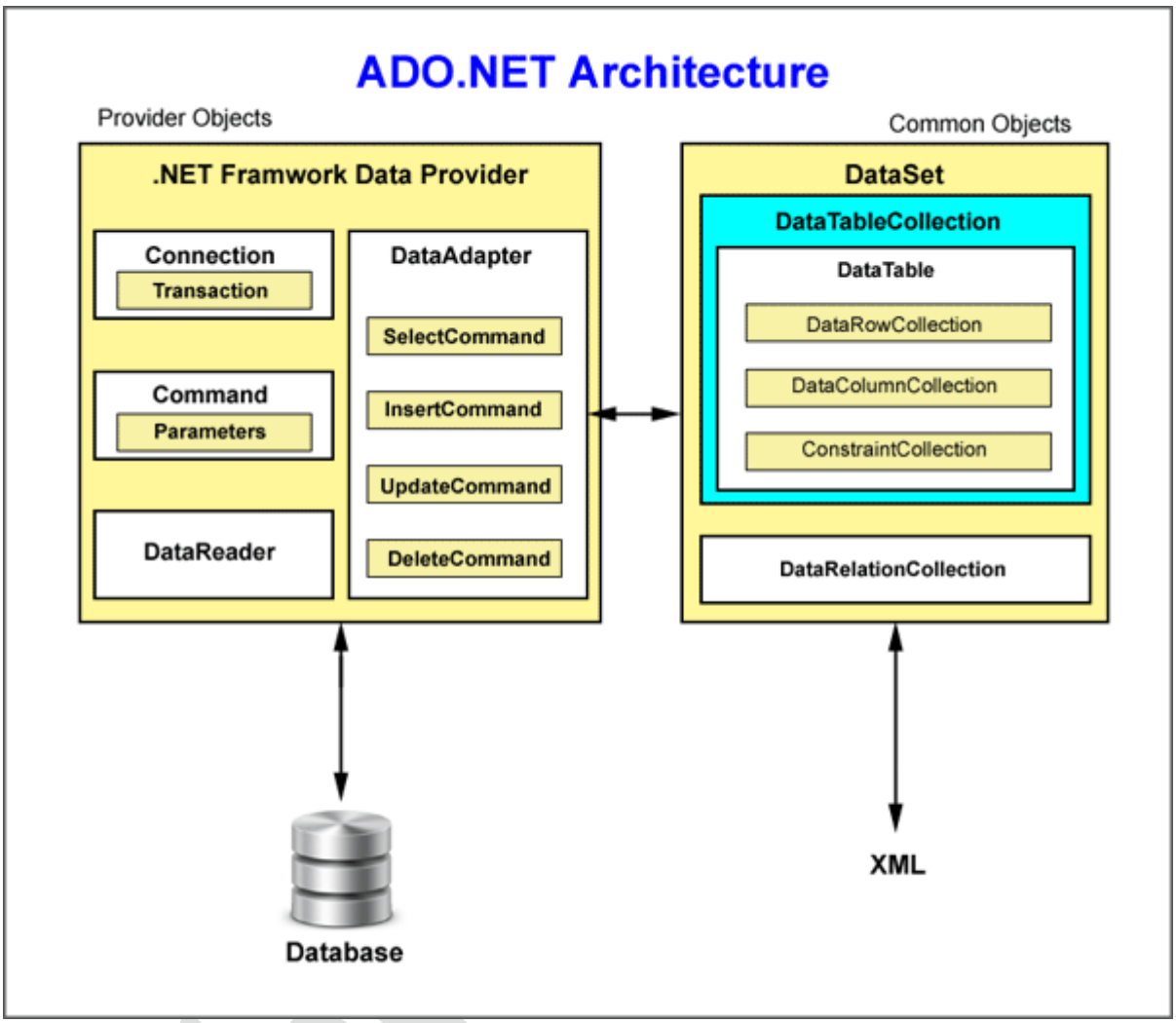
Toegang tot alle databanken door availability van de driver.

3&4. Pure java

Directe communicatie met de database.

Werkwijze :

1. Laden van de juiste driver
2. Maken van een connectie
3. Maken van een statement-object
4. Uitvoeren queries
5. Sluiten van de connectie



Doelstellingen :

multi-tier:
gemakkelijk code op verschillende lagen van toepassing gebruiken

disconnected:
enkel tijdelijke connecties naar databanken

XML-gebaseerd:
voor data-tansport tussen lagen
relationale data <-> XML-data

Schaalbaar

Snel:
vooral in combinatie met een SQLServer

Nadelen:

Niet mogelijk om van databankprovider te wijzigen zonder code te wijzigen

performantie
wijzigingen beperkt
geen wijzigingen binnen ODBC

Native databankproviders maar voor beperkte aantal databanken beschikbaar

Data Provider

Brug tussen toepassing en databron (databank)

Connection:
Verbinding met databank

Command:
instructie voor databank (typisch SQL)

Parameter:
parameters bij command

DataReader:
Snel ophalen van data, enkel voorwaarts doorlopen van een resultaatset

DataAdapter:
Verbinding tussen dataprovider en DataSet, zorgt dat DataSet gevuld wordt door query

CommandBuilder:
als wijzigingen in DataSet terug naar de databank moeten

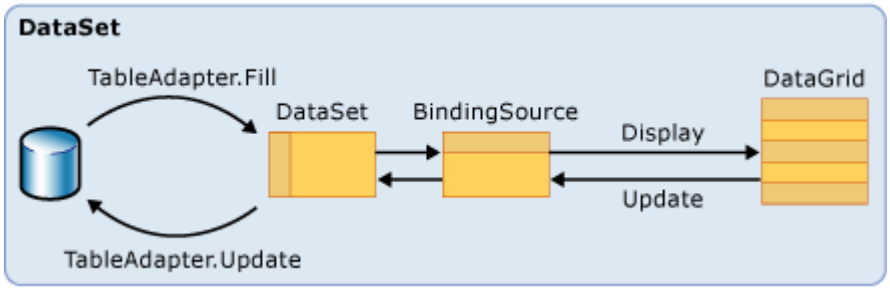
DataSet

Kan meerdere DataTables bevatten

Onafhankelijke datacontainer

Vullen d.m.v. databankquery
Manueel vullen

Wijzigingen aan DataSet worden bijgehouden



Ondervragen mogelijk zonder Database

Gemakkelijk converteerbaar naar XML

Kunnen ook data-integriteit verzorgen

ORM (Object Relational Mapping)

Programma: objecten
Databank: Relationeel

Niet bezighouden met data uit resultSet te halen en in objecten te plaatsen, of expliciet de data in de objecten in databank te bewaren.

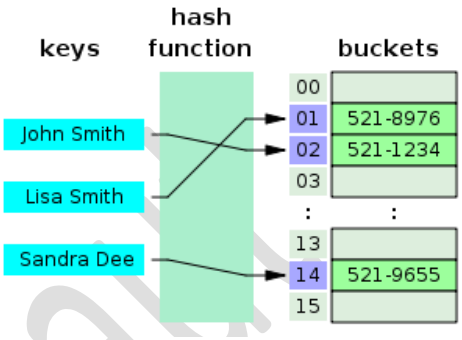
Definiëren van mapping (vertaling) tussen klassen en tabellen
Relaties bijplaatsen

Het grote verschil:
Geen overerving bij relationele databanken.
Oplossing:
Verschillende tabellen maken en de tabellen linken met elkaar via een sleutel.
Zoals bijvoorbeeld:
Persoon <-> Student
Persoon <-> Docent

Basis datastructuren, algoritmen en programma-optimalisatie

Meest gebruikte datastructuren

- Array
- Gelinkte lijst
- Associatieve array/map/dictionary:
  d.m.v. een sleutelobject een ander object benaderen
- Hashtabel:
  hash van sleutel -> index
- Set:
  verzamelingen (niet manipuleren)
- Bomen, binaire bomen, gebalanceerde bomen, heap
- grafen



Hashtabel

Keuze baseren op:

- Functionaliteit
- Snelheid
- Verbruikt geheugen

Oefeningen:

Veelterm, schaarse veelterm

We slaan de veelterm op in een array de index van de array geeft de graad weer en in de array staat dan de respectievelijke coëfficiënt.

Bij een schaarse veelterm –waarbij veel nullen voorkomen- moeten we echter de macht en grondgetal opslaan –in een array-.

Schaarse matrix

Bij een schaarse matrix moeten we het aantal kolommen en rijen specifiëren.
We moeten ook het aantal niet nul rijen vermelden alsook de getallen.

#rijen	Rij	…	…			
#kolommen	Kolom	…	…			
#niet-nulrijen	Getal	…	…			

Onderdriehoeksmatrix

Als we enkel de (niet-nul)elementen in een array plaatsen dan moet onze array k elementen lang zijn.

$$1 + 2 + \cdots + (n-1) + n$$

maar vermits dit moeilijk in formule te vatten is tellen we
$n + (n-1) + \cdots + 2 + 1$ erbij op en delen we door 2:

$$k = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Als we nu willen weten waar het element k_{ij} zich bevindt in de array (index):
(we houden er rekening mee dat i en j beginnen vanaf 0)

$$\underbrace{\frac{i(i+1)}{2}}_{\#elementen\ voor\ rij\ i} + \underbrace{(j+1)}_{\substack{aantal\ elementen\ in\ rij\ i \\ tot\ aan\ element\ k_{ij}}} \underbrace{-1}_{\substack{zodat\ index\ gegeven\ wordt \\ en\ niet\ de\ plaats}}$$

$$L = \begin{bmatrix} l_{1,1} & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ l_{2,1} & l_{2,2} & 0 & \cdots & \cdots & 0 & 0 \\ l_{3,1} & l_{3,2} & l_{3,3} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & 0 & 0 \\ l_{n-1,1} & l_{n-1,2} & l_{n-1,3} & \cdots & \cdots & l_{n-1,n-1} & 0 \\ l_{n,1} & l_{n,2} & l_{n,3} & \cdots & \cdots & l_{n,n-1} & l_{n,n} \end{bmatrix}$$

Algoritme

Algoritme is een eindige lijst van goed-gedefinieerde instructies om op een effectieve manier een bepaald doel te bereiken.
Met:

- Een start-toestand
- 0 of meerder inputs
- Minstens 1 output/Resultaat
- Eindtoestand (niet oneindig lang rekenen)

Opmerking een algoritme is niet noodzakelijk deterministisch[1], het kan ook randomized zijn.

De beoordeling van een algoritme, op:

- De correctheid (soms benaderingen)
- De snelheid (time-complexity)
- Het verbruikte geheugen (space-complexity)
- Vaak time-space trade-off

Snelheid

- Meten maar…
  - Afhankelijk van computer, programmeertaal, compiler, programmeervaardigheid
  - Implementeren (tijd) en uitvoeren (100jaar)
- Schatten
  - Tellen aantal opdrachten en aantal keren uit te voeren
  - Big O notatie:
    functie van inputgrootte (# gegevens te verwerken)

De Big-O-notatie wordt gebruikt om de (tijds)complexiteitsgraad van algoritmen uit te drukken. Deze notatie is een 'slechtste-geval' maat. Ze geeft enkel informatie over het maximaal aantal elementaire bewerkingen dat een algoritme bij gegeven invoergrootte zal uitvoeren. De notatie is machine-onafhankelijk.

- O(1)
- O(log(n))
- O(n)
- O(nlog(n))
- O(n^2)
- O(n^3)
- polynomiaal O(n^k)
- exponentieel

Informeel komt het erop neer dat een algoritme A bij gegeven invoergrootte n (groot genoeg) steeds minder dan f(n) elementaire bewerkingen zal uitvoeren.

[1] Met dezelfde input(s) dezelfde output(s) bereiken

Optimaliseren van software

Ontwerp

- ontwerp architectuur (hoe data benaderen, waar zit de data)
- ontwerp/keuze algoritme

Source code

- keuze datastructuren (geheugen/snelheid)
- goede code (geheugen/snelheid)
- opgelet: optimizing compilers

Build level

- Preprocessor verwijderen onnodige software
- Optimaliseren voor specifieke hardware

Run-time optimalisatie

(code past zichzelf aan, aan reeds gegeven input)

Paretoprincipe:
20% van de operaties gebruiken 80% van de resources.
In software:
90% van de uitvoeringstijd door 10% code uit te voeren.

Dus eerst profiling!

- Arithmetic operation performance is ordered roughly by: transcendental functions, square root, modulo, divide, multiply, add/subtract/mutiply by power of 2/divide by power of 2/modulo by power of 2.
- resultaten van (tussen)berekeningen bijhouden ipv elke keer opnieuw te berekenen (meestal in tabellen)
- maar grote tabellen meestal niet gecached → toch traag
- Control flow is ordered roughly by: indirect function calls, switch() statements, fixed function calls, if() statements, while() statements.
- resultaten elk stadium berekend door complexe berekeningen als incrementele aanpak veel sneller is of omgekeerd?

Grondige optimalisatie verstoort meestal leesbaarheid programma dus enkel als nodig (wel altijd de basis doen)

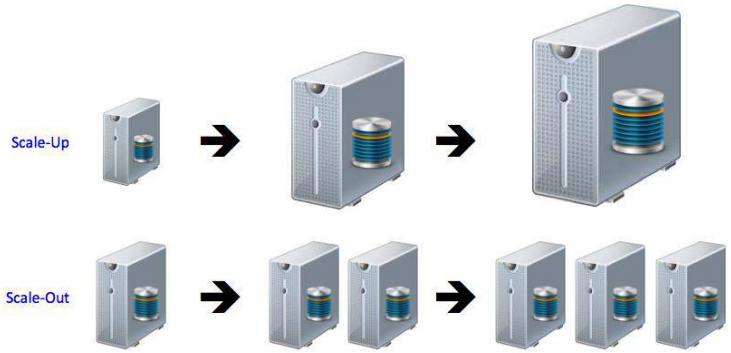
Opgelet, werk optimaliserende compilers niet tegen

Voor grondige optimalisatie: hou rekening met onderliggende hardware

Data

Begrippen

Scale up & scale out

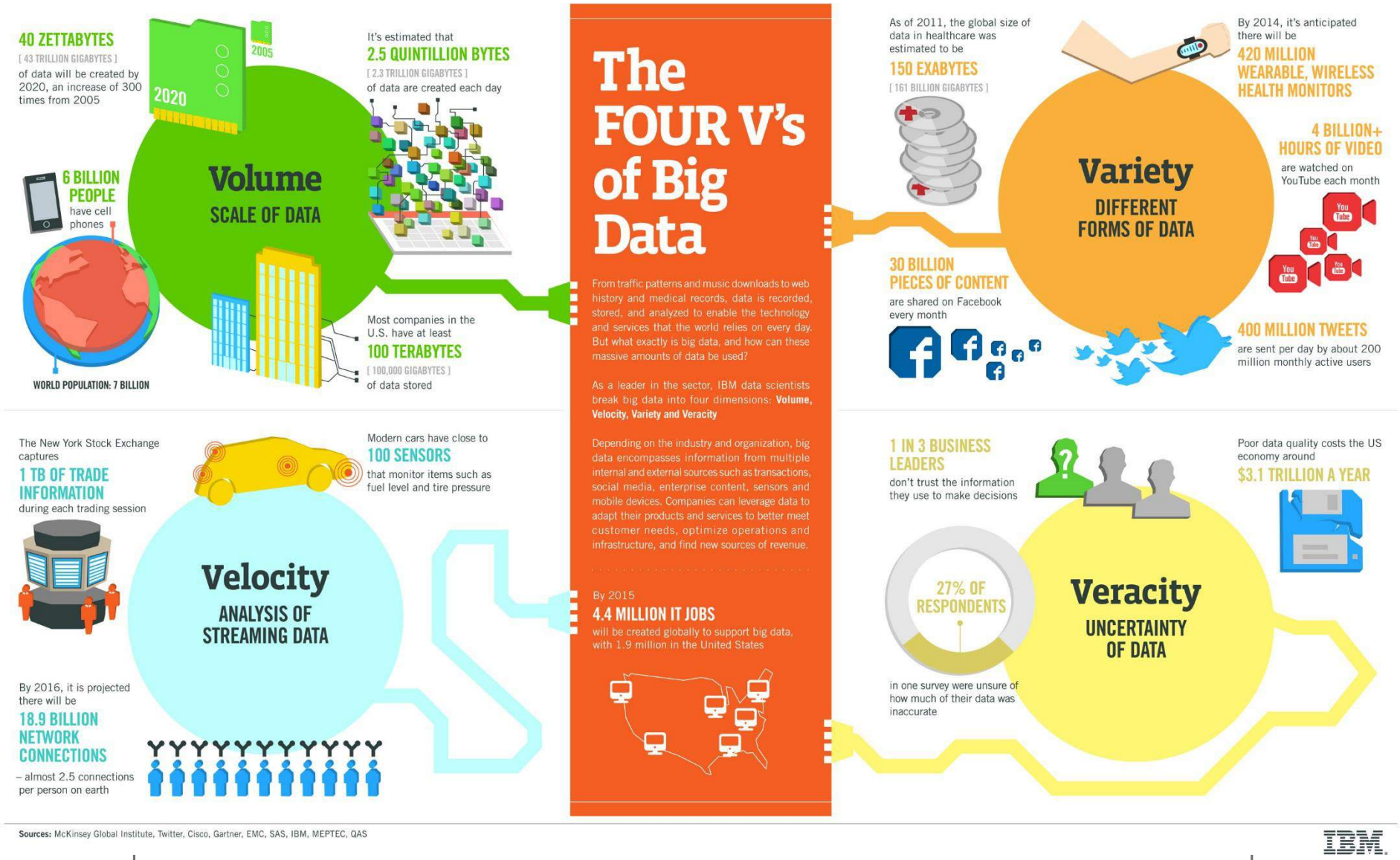


Uitdagingen

- lagere kostprijs
  - licentiekost
  - minder tijd nodig voor ontwerp
    - normalisatie
    - partitionering tabellen
    - index optimalisatie
    - reorganisatie tabellen
  - minder tijd nodig voor administratie
    - tuning hardware
    - tuning operating system
    - tuning database server
    - tuning bufferpool, shared memory area, …
  - opm: bestaande databases: constant tunen, optimaliseren of traag
- snellere query verwerking
  - sub-second response tijden
  - meer gebruikers
  - verschillende groepen van gebruikers
  - meer queries
  - meer complexe queries
  - meer queries op gedetailleerde data
- toenemende flexibiliteit
  - meer tabellen, meer kolommen
  - wijzigen bestaande structuur
  - wijzigingen SQL-databases: zeer duur (alles wijzigen)
  - indexering
  - partitionering
  - buffering
- meer data

Big data voor Analytics

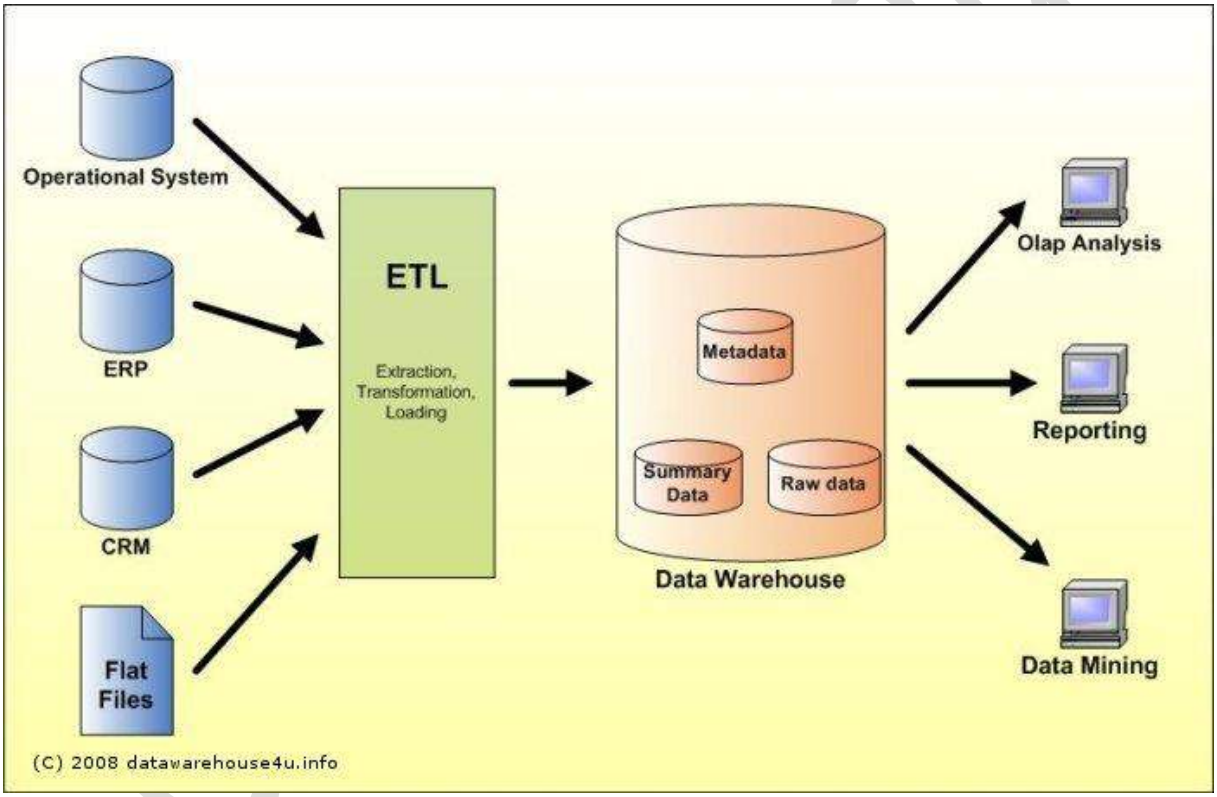
- overal sensoren (productie, autowegen, …)
- klantendata → profielen
    - bv kleine boekenwinkel, regelmatige klant → eigenaar weet wat je graag leest. miljoenen klanten (Amazon)????
    - bv gokken in Las Vegas: wanneer moet ik klant gratis kaarten voor show aanbieden zodat hij blijft gokken?
    - minder voedselverspilling (verwachten goed weer → zoveel barbecue vlees, sla, …)
- fraude detectie door social media te scannen
  (thuis wegens rugpijn en publiceert foto's deelname marathon)
- NSA
- alle grote schepen worden getraced van ver (vanaf Suez-kanaal) om binnenkomen in haven te optimaliseren


40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005
2005
2020
Volume
SCALE OF DATA
It's estimated that
2.5 QUINTILLION BYTES
[2.3 TRILLION GIGABYTES]
of data are created each day
6 BILLION PEOPLE
have cell phones
WORLD POPULATION: 7 BILLION
Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored
The Four V's of Big Data
From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?
As a leader in the sector, IBM data scientists break big data into four dimensions: Volume, Velocity, Variety and Veracity
Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.
By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States
As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]
30 BILLION PIECES OF CONTENT
are shared on Facebook every month
Variety
DIFFERENT FORMS OF DATA
By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS
4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month
400 MILLION TWEETS
are sent per day by about 200 million monthly active users
The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session
Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure
Velocity
ANALYSIS OF STREAMING DATA
By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth
1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions
Poor data quality costs the US economy around
$3.1 TRILLION A YEAR
27% OF RESPONDENTS
in one survey were unsure of how much of their data was inaccurate
Veracity
UNCERTAINTY OF DATA
Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS
IBM.

Data Warehousing

Een **datawarehouse** (vaak afgekort tot DWH) is een gegevensverzameling die in een dusdanige vorm is gebracht dat terugkerende en ad-hoc vragen relatief in korte tijd beantwoord kunnen worden zonder dat de bronsystemen zelf daardoor overmatig belast worden. Hierin onderscheidt een datawarehouse zich van een standaard database. De betreffende gegevens zijn afkomstig van en worden op geautomatiseerde wijze onttrokken aan de bronsystemen. Gegevens kunnen in een datawarehouse niet worden ingevoerd of aangepast door gebruikers zelf. Als aanpassing van gegevens nodig is dan dient dit òf in de bronsystemen òf in het ETL proces (door aanpassing van de daarin vastgelegde regels) plaats te vinden.

- data mining
- marktonderzoek
- decision support
- statistieken



Analytical Database Servers

Kolomgebaseerde RDB's

A columnar database is a database management system (DBMS) that stores data in columns instead of rows.

The goal of a columnar database is to efficiently write and read data to and from hard disk storage in order to speed up the time it takes to return a query.

In a columnar database, all the column 1 values are physically together, followed by all the column 2 values, etc. The data is stored in record order, so the 100th entry for column 1 and the 100th entry for column 2 belong to the same input record. This allows individual data elements, such as customer name for instance, to be accessed in columns as a group, rather than individually row-by-row.

Here is an example of a simple database table with 4 columns and 3 rows.

ID	Last	First	Bonus
1	Doe	John	8000
2	Smith	Jane	4000
3	Beck	Sam	1000

In a row-oriented database management system, the data would be stored like this: 1,Doe,John,8000;2,Smith,Jane,4000;3,Beck,Sam,1000;

In a column-oriented database management system, the data would be stored like this: 1,2,3;Doe,Smith,Beck;John,Jane,Sam;8000,4000,1000;

One of the main benefits of a columnar database is that data can be highly compressed. The compression permits columnar operations — like MIN, MAX, SUM, COUNT and AVG— to be performed very rapidly. Another benefit is that because a column-based DBMSs is self-indexing, it uses less disk space than a relational database management system (RDBMS) containing the same data.

As the use of in-memory analytics increases, however, the relative benefits of row-oriented vs. column oriented databases may become less important. In-memory analytics is not concerned with efficiently reading and writing data to a hard disk. Instead, it allows data to be queried in random access memory (RAM).

In-memory RDB's

In-memory analytics is an approach to querying data when it resides in a computer's random access memory (RAM), as opposed to querying data that is stored on physical disks. This results in vastly shortened query response times, allowing business intelligence (BI) and analytic applications to support faster business decisions.

As the cost of RAM declines, in-memory analytics is becoming feasible for many businesses. BI and analytic applications have long supported caching data in RAM, but older 32-bit operating systems provided only 4 GB of addressable memory. Newer 64-bit operating systems, with up to 1 terabyte (TB) addressable memory (and perhaps more in the future), have made it possible to cache large volumes of data -- potentially an entire data warehouse or data mart -- in a computer's RAM.

In addition to providing incredibly fast query response times, in-memory analytics can reduce or eliminate the need for data indexing and storing pre-aggregated data in OLAP cubes or aggregate tables. This reduces IT costs and allows faster implementation of BI and analytic applications. It is anticipated that as BI and analytic applications embrace in-memory analytics, traditional data warehouses may eventually be used only for data that is not queried frequently.

Opmerking:

- data ook opslaan (persistency)
- lange startup (data in geheugen laden)
- prijs
- consistentie (hoe data naar persistente opslag)

Data Warehouse appliances (factor 100 sneller)

A data warehouse appliance is an all-in-one "black box" solution optimized for data warehousing. The appliance consists of a server pre-built with operating system, storage, database management system (DBMS), and software.

A data warehouse appliance is a combination hardware and software product that is designed specifically for analytical processing. An appliance allows the purchaser to deploy a high-performance data warehouse right out of the box.

In a traditional data warehouse implementation, the database administrator (DBA) can spend a significant amount of time tuning and putting structures around the data to get the database to perform well for large sets of users. With a data warehouse appliance, however, it is the vendor who is responsible for simplifying the physical database design layer and making sure that the software is tuned for the hardware.

When a traditional data warehouse needs to be scaled out, the administrator needs to migrate all the data to a larger, more robust server. When a data warehouse appliance needs to be scaled out, the appliance can simply be expanded by purchasing additional pug-and-play components.

A data warehouse appliance comes with its own operating system, storage, database management system (DBMS) and software. It uses massively parallel processing (MPP) and distributes data across integrated disk storage, allowing independent processors to query data in parallel with little contention and redundant components to fail gracefully without harming the entire platform. Data warehouse appliances use Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), and OLE DB interfaces to integrate with other extract-transform-load (ETL) tools and business intelligence (BI) or business analytic (BA) applications.

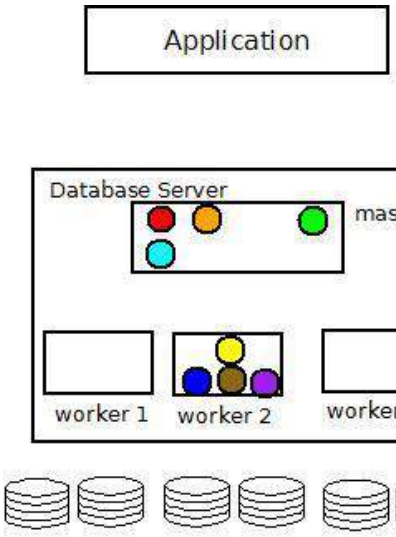
In-database Analytics

In-database analytics is a technology that allows data processing to be conducted within the database by building analytic logic into the database itself. Doing so eliminates the time and effort required to transform data and move it back and forth between a database and a separate analytics application.

An in-database analytics system consists of an enterprise data warehouse (EDW) built on an analytic database platform. Such platforms provide parallel processing, partitioning, scalability and optimization features geared toward analytic functionality.

In-database analytics allows analytical data marts to be consolidated in the enterprise data warehouse. Data retrieval and analysis are much faster and corporate information is more secure because it doesn't leave the EDW. This approach is useful for helping companies make better predictions about future business risks and opportunities, identify trends, and spot anomalies to make informed decisions more efficiently and affordably.

Companies use in-database analytics for applications requiring intensive processing – for example, fraud detection, credit scoring, risk management, trend and pattern recognition, and balanced scorecard analysis. In-database analytics also facilitates ad hoc analysis, allowing business users to create reports that do not already exist or drill deeper into a static report to get details about accounts, transactions, or records.



- parallel: zwaardere operatoren in master
- MapReduced-based architectuur:
    - enkel rode bol (analytical functions) in master
    - 99% geparallelliseerd

SSD (solid State disk)

- SSD drive performance: 160
- potentieel factor 100 winst (direct access), maar bij DB max 10
  - DB: scannen hele schijf ipv direct access want slecht als arm verplaatst moet worden
  - DB: weet niet welke soort disks onder
  - jaren duren on DB-code aan te passen

NOSQL

A NoSQL or Not Only SQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability. The data structure (e.g. key-value, graph, or document) differs from the RDBMS, and therefore some operations are faster in NoSQL and some in RDBMS. There are differences though and the particular suitability of a given NoSQL DB depends on the problem to be solved (e.g. does the solution use graph algorithms?)

Document stores

The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, and JSON as well as binary forms like BSON, PDF and Microsoft Office documents (MS Word, Excel, and so on).

Different implementations offer different ways of organizing and/or grouping documents:

- Collections
- Tags
- Non-visible Metadata
- Directory hierarchies

Compared to relational databases, for example, collections could be considered analogous to tables and documents analogous to records. But they are different: every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

Documents are addressed in the database via a unique key that represents that document. One of the other defining characteristics of a document-oriented database is that, beyond the simple key-document (or key–value) lookup that you can use to retrieve a document, the database will offer an API or query language that will allow retrieval of documents based on their contents.

Graph Stores

This kind of database is designed for data whose relations are well represented as a graph (elements interconnected with an undetermined number of relations between them). The kind of data could be social relations, public transport links, road maps or network topologies, for example.

Key-value stores

Key–Value stores use the associative array (also known as a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key–value pairs, such that each possible key appears at most once in the collection.

The key–value model is one of the simplest non-trivial data models, and richer data models are often implemented on top of it. The key–value model can be extended to an ordered model in which keys are maintained in lexicographic order. This extension is powerful in that it allows efficient processing of key ranges.

Key–Value stores can use consistency models ranging from eventual consistency to serializability. Some support ordering of keys. Some maintain data in memory (RAM), while others employ solid-state drives or rotating disks.

Colum-family stores

A column family is a NoSQL object that contains columns of related data. It is a tuple (pair) that consists of a key-value pair, where the key is mapped to a value that is a set of columns. In analogy with relational databases, a column family is as a "table", each key-value pair being a "row".Each column is a tuple (triplet) consisting of a column name, a value, and a timestamp. In a relational database table, this data would be grouped together within a table with other non-related data.