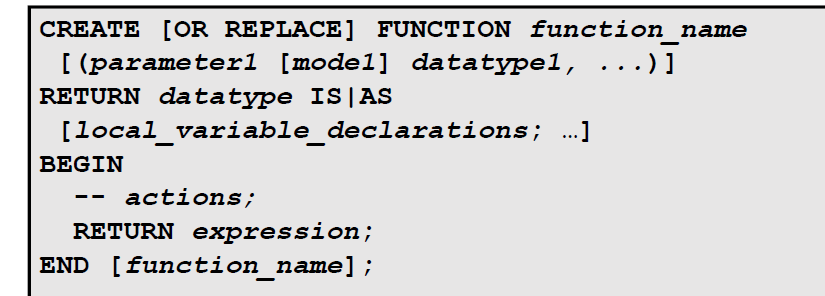
# SQL

# BEGIN voor elke keer opstarten van sqlplus

SET SERVEROUT ON

# Functions

De syntax van een functie is als het volgt



Hier kan je create doen of een bestaane functie vervangen

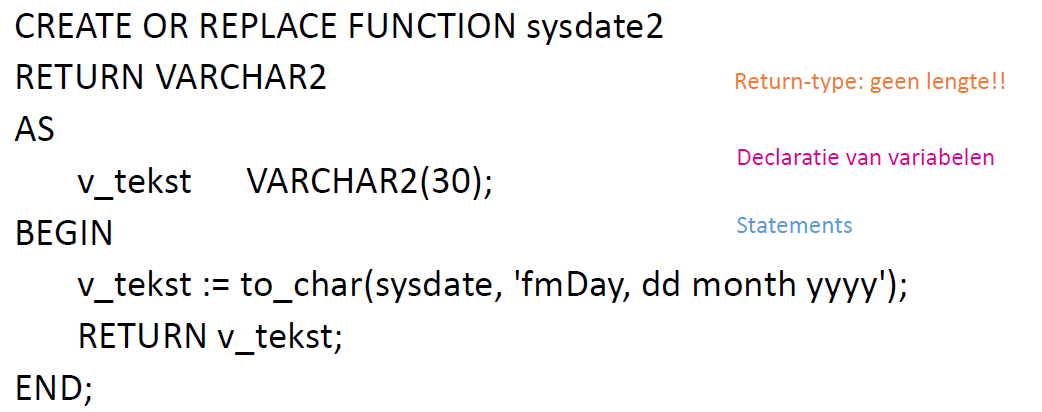
Dan de parameter geef je mee met de naam p\_iets.

Dit komt of uit de database of geef je mee als data.

Daarna zeg je wat er terug moet gegeven worden

Daarna zeg je welke acties moet gebeuren en geef je een waarde terug en zeg je op het einde end

Bv



Elke instructie eindigt met een ;

Elke toekenning gebeurt door :=

Je kan ook andere functies gebeuren door ze toe te kennen aan een andere variabele die he hebt aangemaakt bv v\_lengte := LENGTH (v\_naam)

Commentaar kan getypt worden door of

Door twee liggende streepjes – voor één lijn

of voor meerdere lijnen /\* commentaar \*/

Als er de melding komt created with comilation errors kan je dit eenvoudig opvragen met het commando “show errors”. Deze zegt waar er ergens de fout zit ipv totaal geen idee te hebben. Vaak zit het op de rij zelf of de vorige rij als je bijvoorbeeld een ; vergeten bent.

De voorwaarden van het aanmaken van de naamgeving van een variabele is:

Moet beginnen meet een letter

Mag letters en getallen bevatten

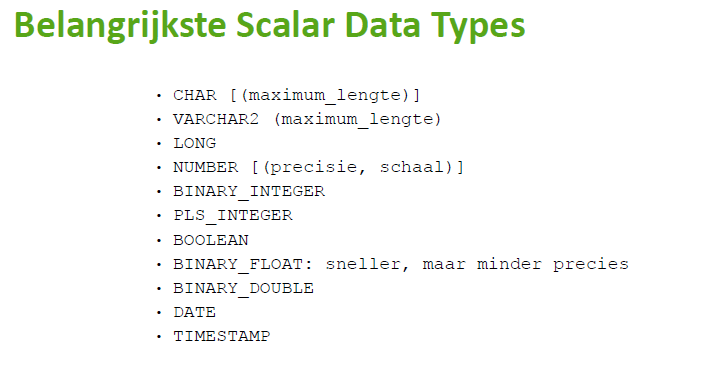
Mag bevatten dollar teken ($), underscore, pond teken(alle andere tekens mag het dus niet bevatten)

Maximale lengte is 30 tekens

Geen gereserveerde woorden

Moet beginnen met v\_

De datatypes die wij gebruiken



## Declaratie met %TYPE

Dit maakt een verwijzing naar een object in de database. Dit zorgt er voor dat het type

Vb:

V\_getal NUMBER(4,1);

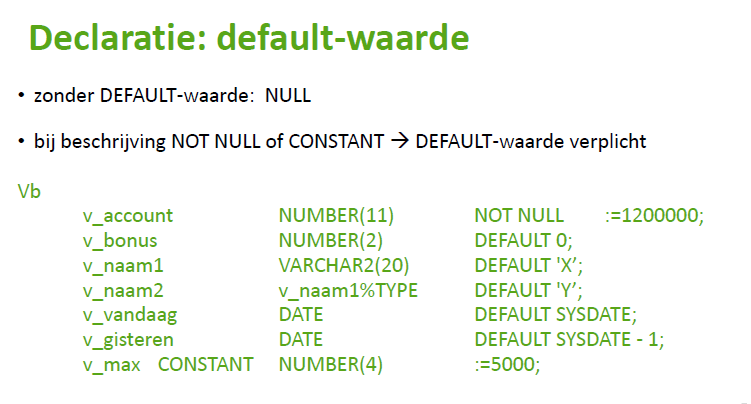
V\_getal2 v\_getal%TYPE;

V\_mndsal employees.salary%TYPE

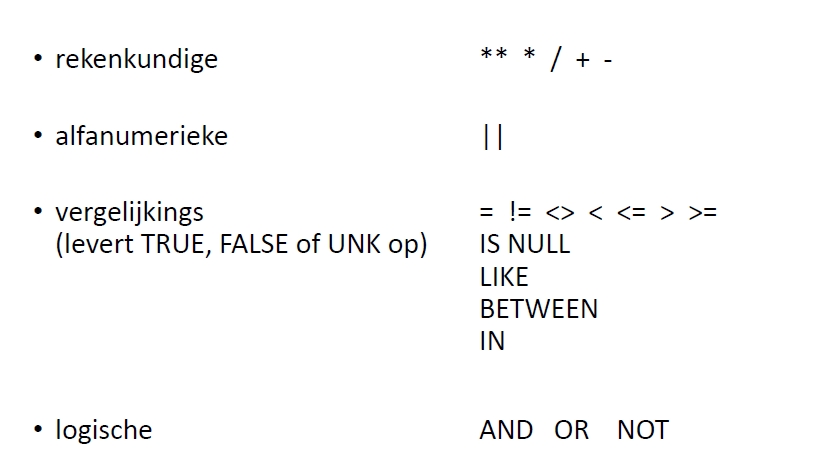
In dit voorbeeld is V\_getal dus een nummer met maximum 4 cijfers en één cijfer na de komma. V\_getal2 is juist het zelfde als V\_getal dus ook een nummer met maximum 4 cijfers en één na de komma.

En V\_mndsal heeft de zelfde waarde als het salary van employees in de database.

Dit is handig als je een varialbe gelijk stelt aan iets uit de database.

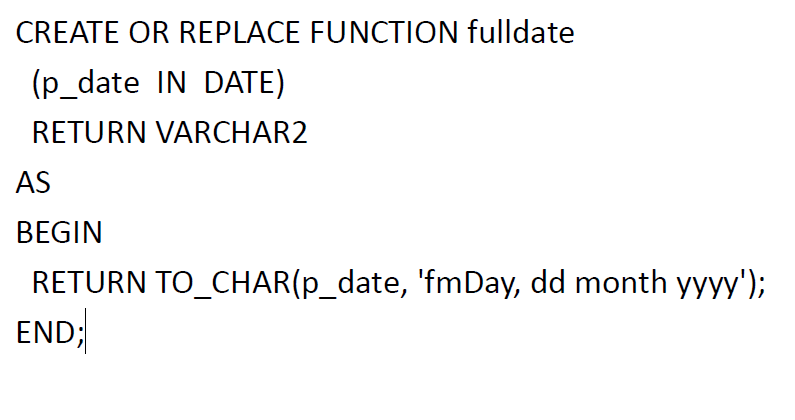


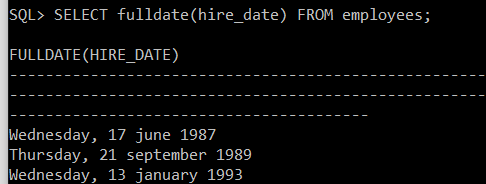
Enkele wiskundige operatoren



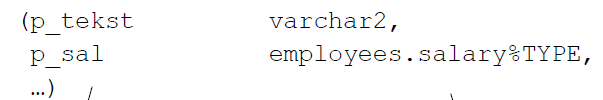
(\*\* nog uit te zoeken wat dit juist doet)

Voorbeeld om een dagen te krijgen in een andere manier uit de database:

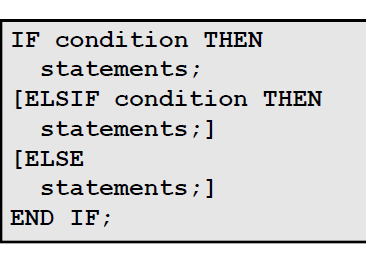




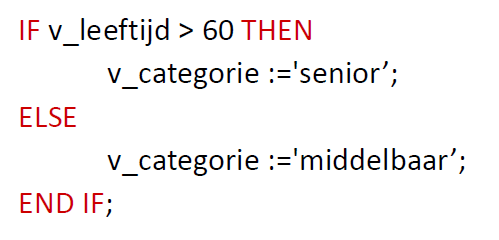
Als je meerdere parameters wilt ingeven kan je dit doen door deze te scheiden met een komma. Dus je schrijft geen twee keer een haakje maar zoals hier onder.



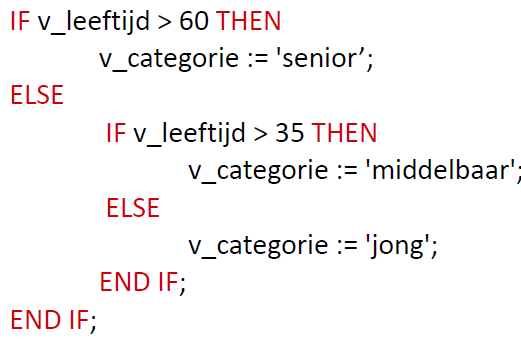
## IF THEN ELSIF ENDIF



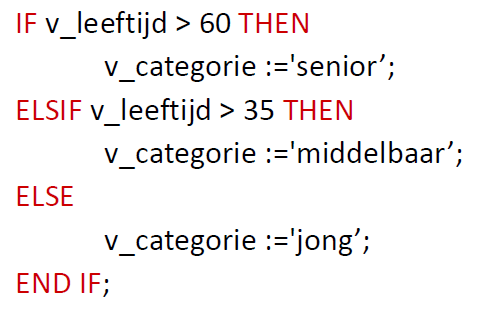
eenvoudig Vb:



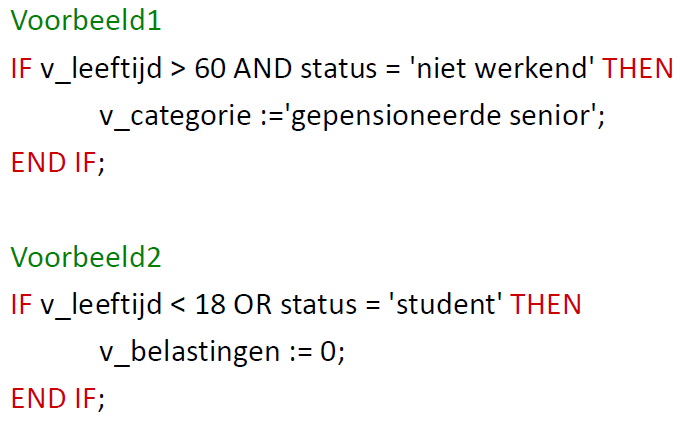
Geneste IF structuur vb:



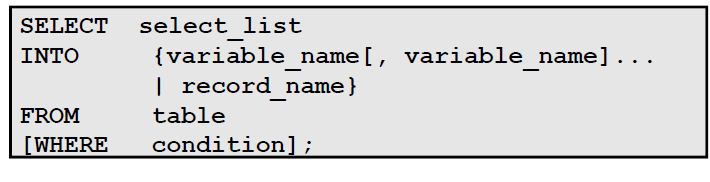
ElsIF structuur vb:



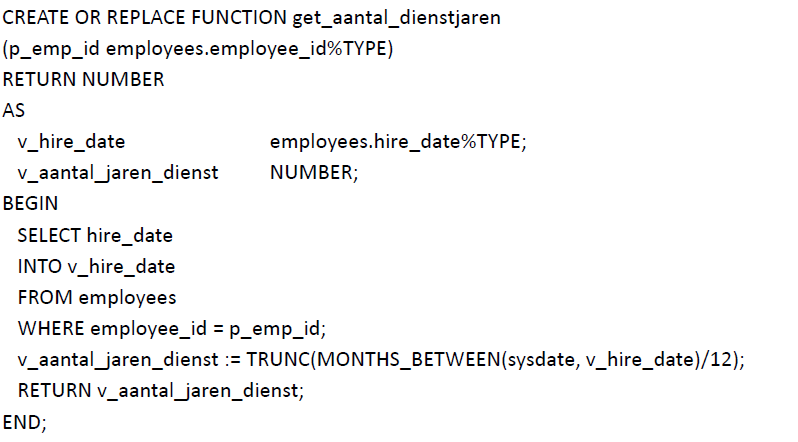
AND en OR structuur vb:



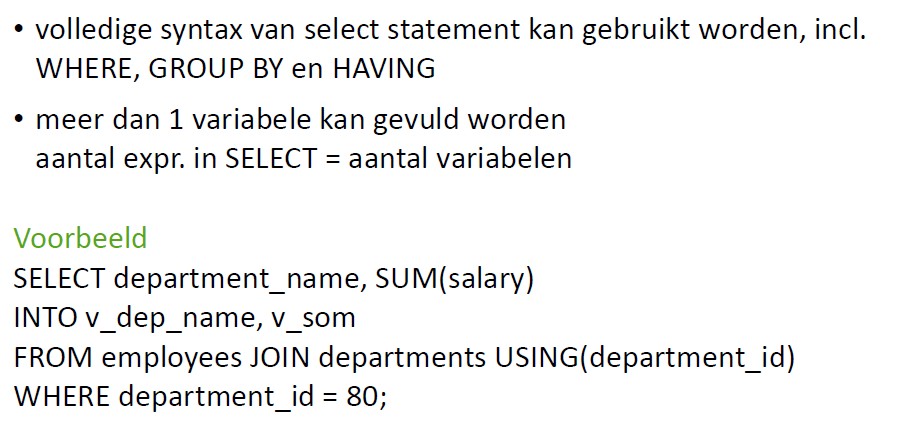
Je kan ook in je functie een SELECT statement gebruiken zodat deze functie iets terug geeft waar deze gelijk aan iets dat je hebt meegeven.



Bv:



Opmerking



Je kan ook een functie verwijderen door het commando

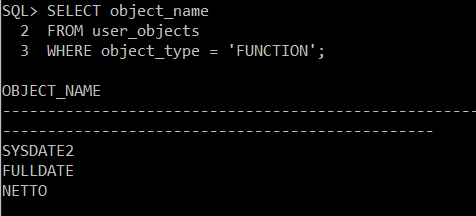
DROP FUNCTION function\_name

Vb

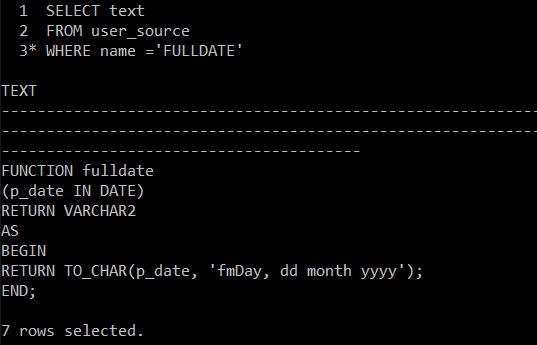
DROP FUNCTION get\_jaarsal;

## Opvragen kenmerken

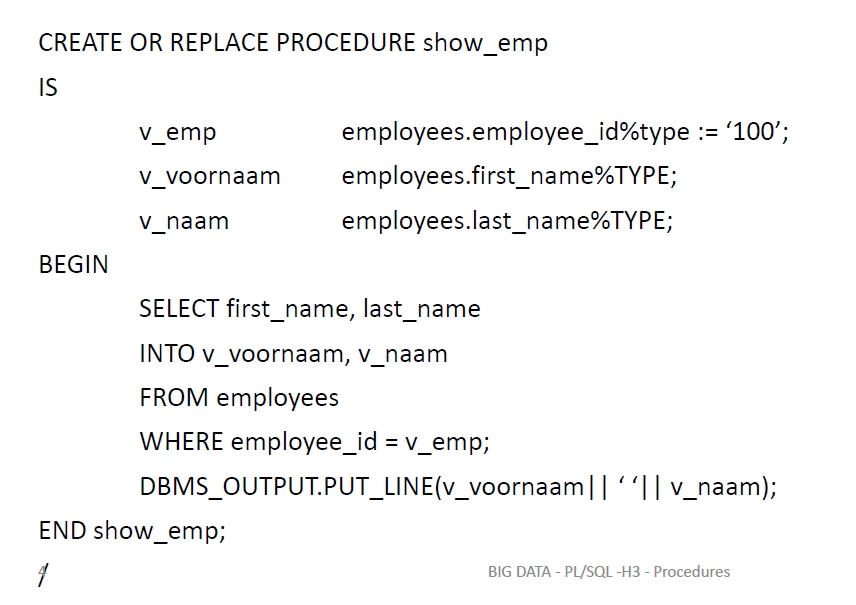
Je kan de functies die geschreven zijn opvragen door het commando



Opvragen van een functie kan je doen door het commando:



# Procedures

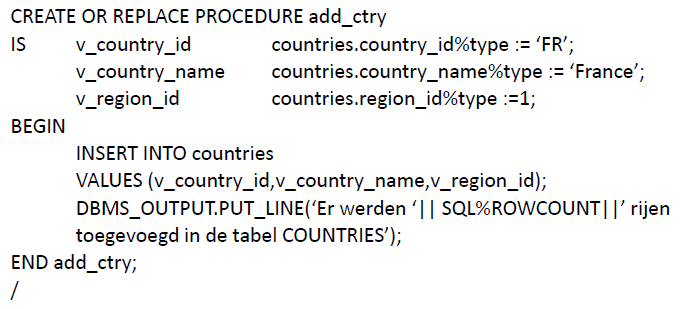


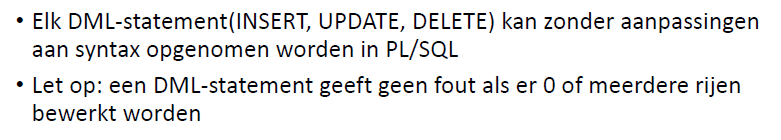
Je kan bij procedures ook alles wat er juist gebeurd in een lijn laten afdrukken door het commando “DBMS\_OUTUT.PUT\_LINE(‘iets’);”

Als je dit wilt gebruiken moet je in je sql plus eerst “SET SERVEROUTPUT ON” typen.

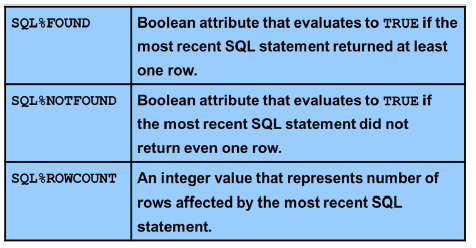
Je kan dit natuurlijk ook automatisch doen door dit commando in LOGIN.SQL te plaatsen

Een voorbeeld zonder parameters

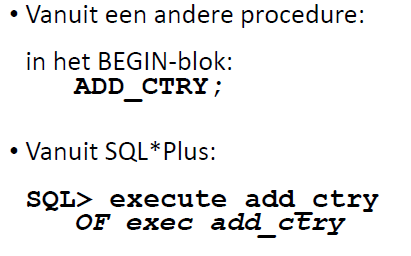




Om dit op te vangen kan je gebruik maken van sommige van volgende functies.

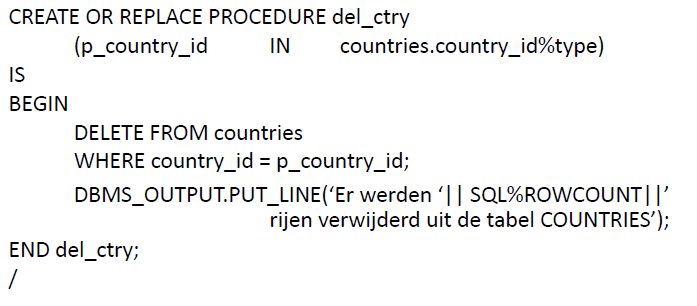


Je kan ook een prdcedure aanroepen in een andere precedure of gewoon in sql plus

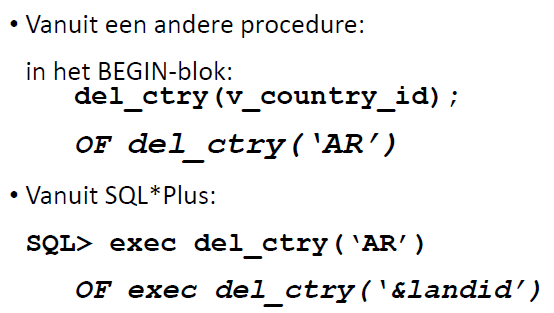


## IN

Als je in een procedure een parameter wilt gebruiken kan je dit gebruiken doen door een parameter mee te geven en het woord IN. Bv.

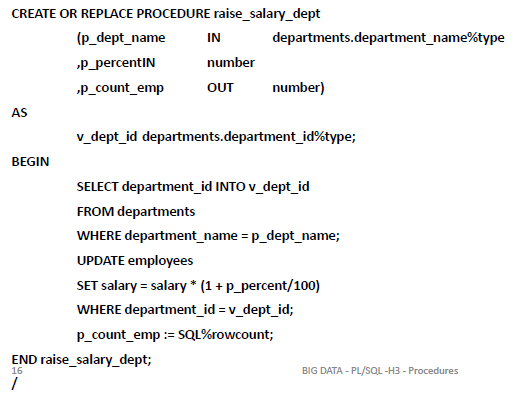


Je kan dan later of in het begin blok of in sql plus direct deze procedure oproepen. Hier geef je dan ook een waarde mee voor deze parameter die je in de procedure geschreven hebt.

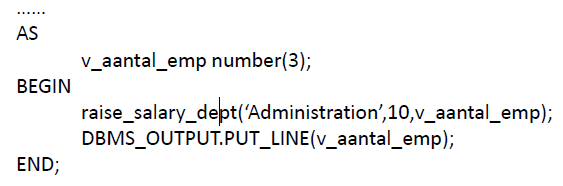


## OUT

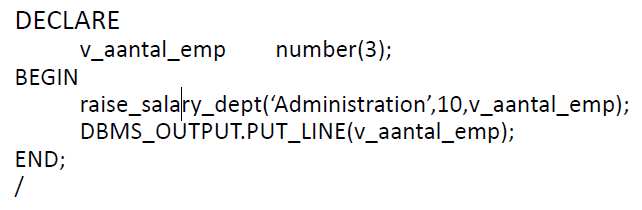
Het in woord gebruik je in sql om iets in te geven in het begin van je functie. Nu kan je ook het woord OUT gebruiken. Dit spreekt redelijk voor zich zelf en is om een woord te outputen op het einde van je procedure.



Hieronder is een voorbeeld dat een andere procedure uitgevoerd word in een andere procedure.

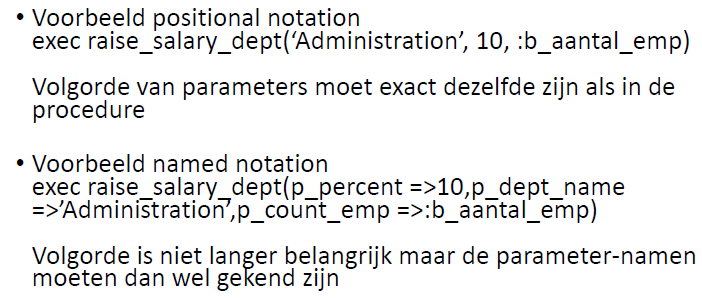


Je kan dit natuurlijk ook doen via een anoniem blok bv.



## Notations

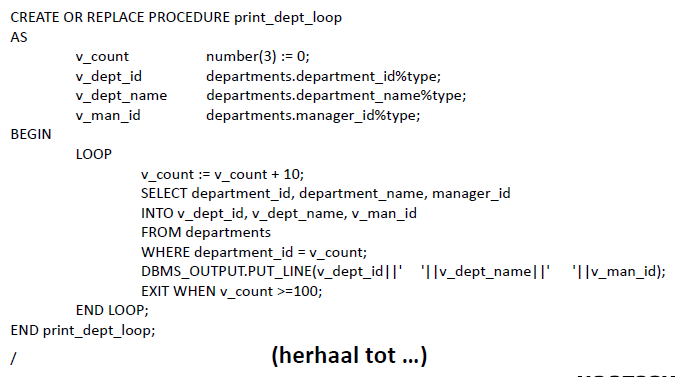
Je kan een procedure op verschillende manieren je informatie meegeven. Je kan dit doen zoals je al lang gewoon ben in de volgorde zoals je die hebt ingegeven of je kan dit doen door named notation. (persoon vind ik dit overbodig om dat je het al lang gewoon bent om in de juiste volgorde te zetten en het maakt je meer typen alleen de volgorde speelt hier geen rol maar kost natuurlijk meer tijd om te typen)



## LOOP

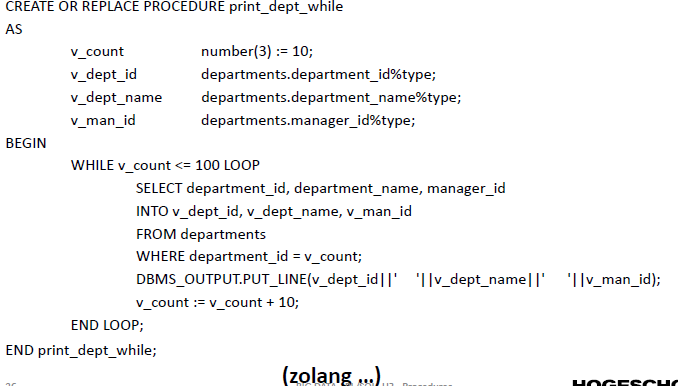
Net als in veel andere programeer talen kan je ook hier met een loop werken. Je kan dit best vergelijken met do while. Je start dit met LOOP en eindigt dit met EXIT WHEN varialbenaam >=< of zo iets dan iets ; en dan END LOOP;

Bv.



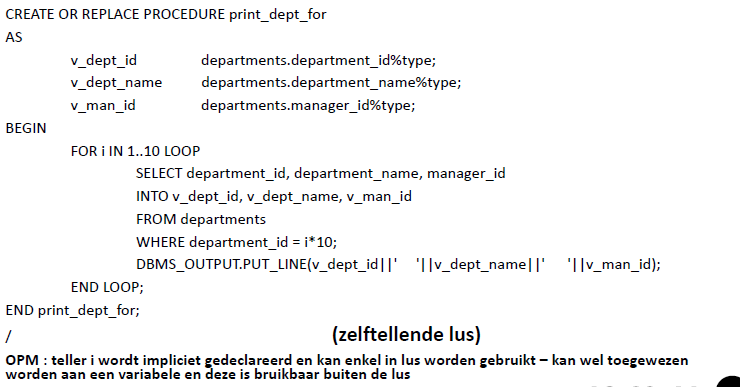
## WHILE

Hetzelfde als bovenstaande voorbeeld maar in plaats van dat er eerst iets gedaan word en dan pas gekeken word of er al aan de voorwaarde voldaan is. Met een while word eerst gekeken of er al aan de voorwaarde voldaan is als dit zo is gaat er niet in de blok anders wel tot er aan voldaan is.



## FOR

Je kan natuurlijk ook een for lus gebruiken dit werkt zoals je het kent in andere programeer talen. Dit telt zelf elke keer plus een van het begin getal dat je hebt ingegeven tot aan het laaste getal dat je hebt ingegeven.



Als je wilt dat je lus met twee verhoogt of met een ander getal dan met een moet je dit doen door ipv gewoon i te pakken kan je dan doen door (i+(i+1)) voor als je i wilt verhogen met twee. Als je dit wilt doen voor bv 3 dan doe je i+i+i+1. Natuurlijk pas je i niet echt aan maar haal je gewoon data uit i en steek dit in een variabel. Natuurlijk moet je dan wel bij de lus ook wat aanpassen. Je moet dit beginnen met BEGINGETAL-1..(AANTALKEREN/AANTALPERSPRONG JE WILT)-1.

Dit zou dus willen zeggen dat het zelfde voorbeeld zo word

CREATE OR REPLACE PROCEDURE print\_dept\_for

AS

v\_dept\_id departments.department\_id%type;

v\_dept\_name departments.department\_name%type;

v\_man\_id departments.manager\_id%type;

BEGIN

FOR i IN 0..4 LOOP

SELECT department\_id, department\_name, manager\_id

INTO v\_dept\_id, v\_dept\_name, v\_man\_id

FROM departments

WHERE department\_id = (i+i+1)\*10;

DBMS\_OUTPUT.PUT\_LINE(v\_dept\_id||' '||v\_dept\_name||' '||v\_man\_id);

END LOOP;

END print\_dept\_for;

/

## Alternative FOR

CREATE OR REPLACE PROCEDURE print\_dept\_cursorloop

AS

BEGIN

FOR rec IN (SELECT department\_id, department\_name, manager\_id

FROM departments

WHERE department\_id between 10 and 100)

LOOP

DBMS\_OUTPUT.PUT\_LINE(rec.department\_id||'

'||rec.department\_name||' '||rec.manager\_id);

END LOOP;

END print\_dept\_cursorloop;

/

## Procedures verwijderen

Net als bij functies kan je ook een procedure verwijderen. Dit doe je door bv.

DROP PROCEDURE raise\_salary\_dept;

## Opvragen kenmerken

Net als bij functies kan je ook dit doen met een procedure. Dit doe je door

**SELECT object\_name**

**FROM user\_objects**

**WHERE object\_type = ‘PROCEDURE';**

Zo kan je alle Prcedures opvragen.

Als je de fucntie van een proceduere wilt opvragen kan je dit doen door dit commando.

SELECT text

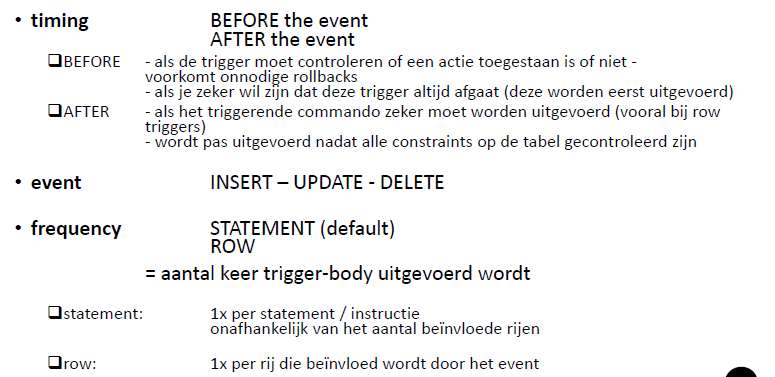
FROM user\_source

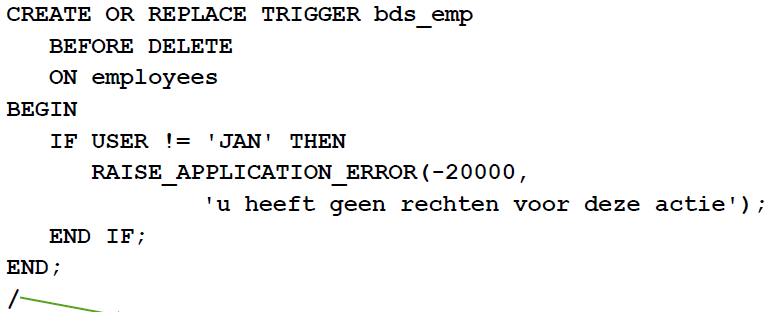
WHERE name = ‘RAISE\_SALARY\_DEPT';

Dit moet altijd met hoofdletters.

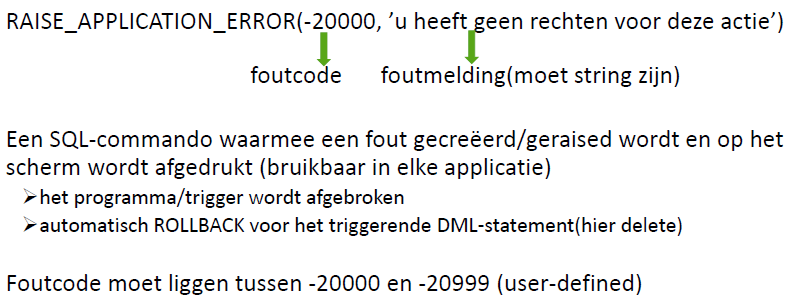
# Database triggers

Als je een trigger maakt dan word er een timing een event en hoevaak deze dit moet uitgevoerd worden bv.





## Foutmelding



## Naamgeving

B=before

A=after

I=insert

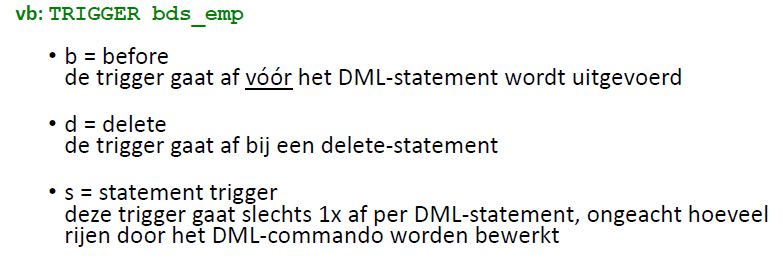
U=Update

D=Delete

S=Statement

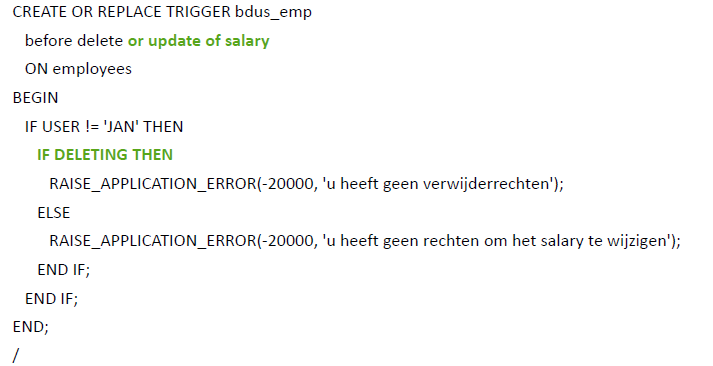
R=Row

Het eerste en het laatste kan je maar een nemen. Maar van de middelste blok kan je meer tegelijk in een trigger.



## Multi statement trigger

Je kan natuurlijk ook meerdere triggers in een schrijven dit kan je doen door bv.



Zoals je hierboven ziet staat hier DELETING. Er zijn 3 van deze functies:

INSERTING

DELETING

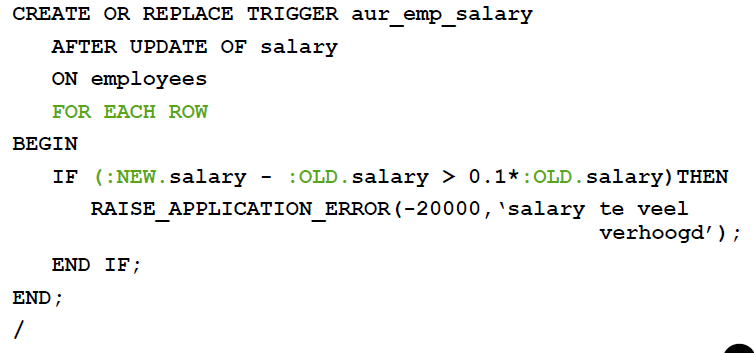
UPDATING

In dit laaste geval kan er ook nog een extra parameter dus bv.

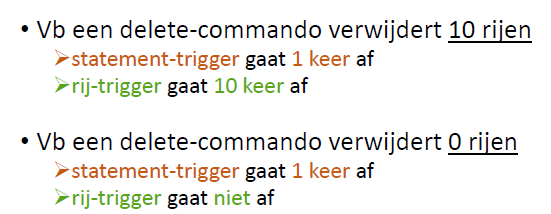
IF updating(‘salary’) THEN

## Row Trigger

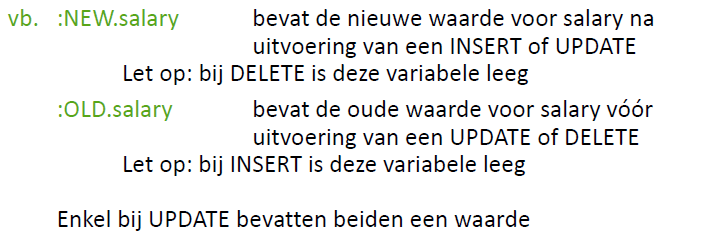
In plaats van een statment kan je ook een row trigger hebben bv.



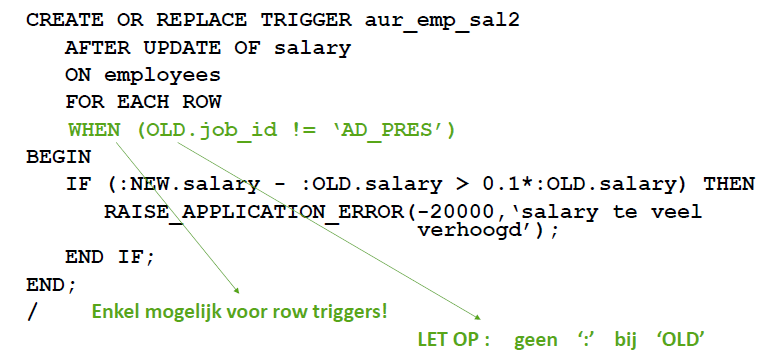
Het grote verschil tussen een statement trigger en een rij trigger is dat een statement 1 keer runt en een rij trigger zo vaak triggert als er rijen worden gemanipuleerd(d/u/i) bv.



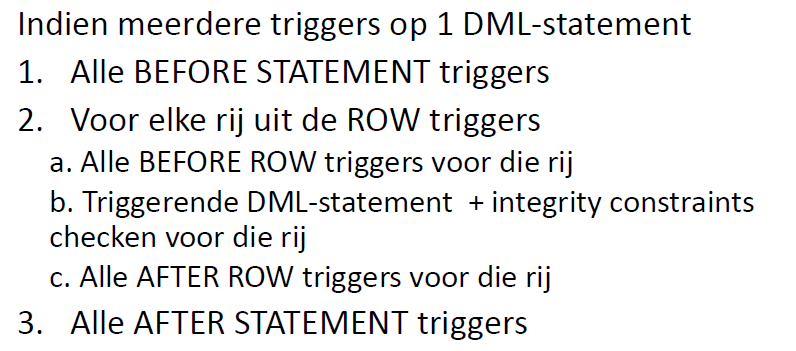
Hier kan je ook de keywords :NEW.kolomnaam en :OLD.kolomnaam dit spreekt denk ik wel redelijk voor zich zelf.



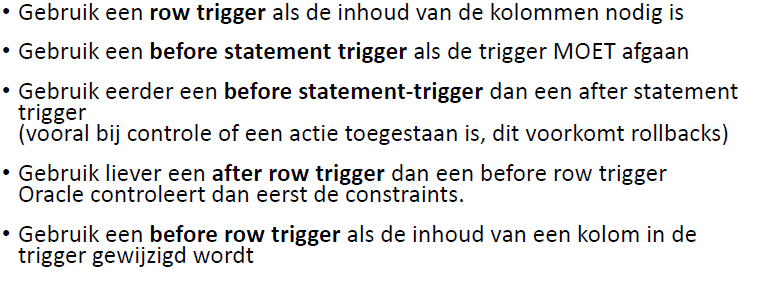
Je kan ook dit uitbreiden met een when. Hierboven zeg je gewoon op elke lijn. Maar je kan ook het woord WHEN gebruiken. Dus stel waar de vorige job\_id niet gelijk is aan iets dan voer je iets uit dus zoals in dit voorbeeld.



## Volgorde van triggers



## Welke trigger moet je kiezen in welke situatie



## Opvragen kenmerken

Net als bij functies en Procedures kan je ook dit doen met een trigger. Dit doe je door

**SELECT object\_name, created, status**

**FROM user\_objects**

**WHERE object\_type = ‘TRIGGER';**

Zo kan je alle Triggers opvragen en kijken of deze actief staan.

Als je de fucntie van een trigger wilt opvragen kan je dit doen door dit commando.

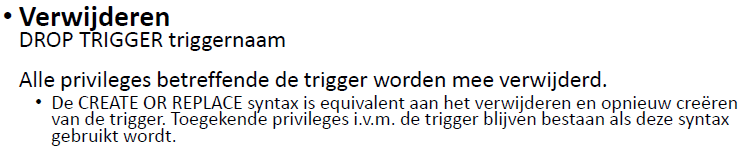
SELECT line,text

FROM user\_source

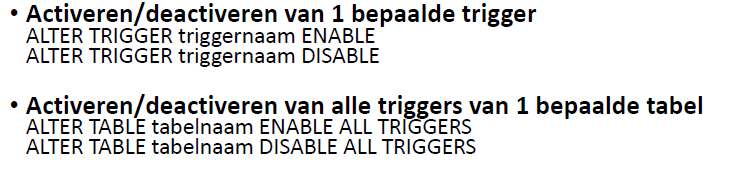
WHERE name = ‘AUR\_EMP\_SALARY';

Dit moet altijd met hoofdletters.

## Verwijderen van een trigger

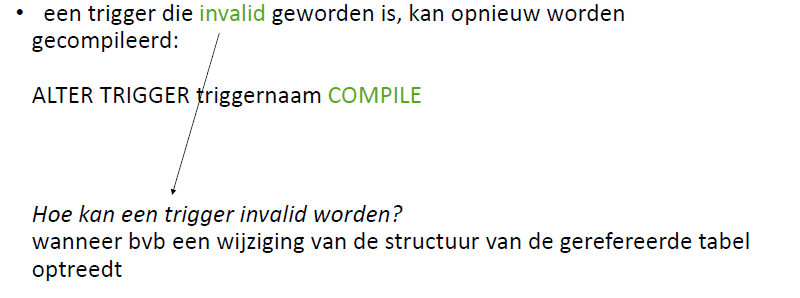


## Activeren/deactiveren van triggers



Natuurlijk standaard staan triggers aan.

Een trigger kan ook invalid worden als de structuur van een databas aangepast word. Dit pas je dan zo aan.



# NO SQL

## Veranderden van database waar je in het werken bent

use naamvandatabase

bv use test

als je dit commando intypt ga je werken met de database test als deze nog niet bestaat zal deze ook aangemaakt worden met dit commando.

## Tonen van alle databasen

show dbs

hier ga je enkel de databasen zien waar al reeds iets inzit

## Tonen van de huidige database

Db

## Tonen van alle collections

Show collections

## Aanmaken van een collection in een database

Omdat het big data is en dus niet als sql is kan elk record anders zijn dit collections die je moet aanmaken en daar een record in steken dit doe je door

db.createCollection(naamvancolletion)

bv

db.createCollection(“dbtest”)

dit word altijd gedaan in de huidige db waar je in aan het werken bent

# Insert

## Aanmaken van een record in een collection

Je kan dan ook een record invoegen in een collection dit kan je doen op veel verschillende manieren de eenvoudige manier bv:

db.studentenregatta.insert({\_id: '001', team: 'PXL', aantal\_likes:820, teamleden: 9})

Je kan best altijd een \_id meegeven omdat anders mongodb zelf een gegenereerde key aanmaken

## Als je meerdere records tegelijk wilt toevoegen kan je dit doen door [] te gebruiken bv:

db.studentenregatta.insert([{\_id: '002', team: 'UCLL', teamleden:8}, {\_id:'003', team 'Uhasselt', aantal\_likes: 400}])

# Update

## Een record aanpassen

Dit kan je op verschillende manieren opvatten of een naam van iets veranderen bv:

db.studentenregatta.update({team:'PXL'},{$set:{team: 'Hogeschool PXL'}})

## een record aanpassen met verschillende records

db.films.update({naam:'Iedreen beroemd'},{$set:{acteurs:['Josse De Pauw', 'Eva van Der Gucht','Werner De Smedt']}}})

## een record aanpassen met één extra waarde te willen toevoegen

als je een extra persoon bv wilt toevoegen aan acteurs moet je elke keer alle acteurs toevoegen omdat dit anders denkt dat die je niet schrijft weg moeten.

db.film.update({naam:'Loft'},$set:{acteurs:['Koen De bouw', 'Filip Peeters', 'Matthisas Schenaeerts']}})

## een record aanpappen met één extra waarde te willen toevoegen met $push

Als je dit niet wilt dit elke keer dit wilt typen kan je ook $push gebruiken dit commando zet een waarde bij een set.

Dus bv:

db.film.update({naam:’Loft’},{$push:{acteurs:’naam4’}})

zo kan je eenvoudig één extra record toevoegen ipv een hele lijst typen bij $set

## of aan een record een veld toevoegen. Bv:

db.studentenregatta.update({team:'Hogeschool PXL'}, {$set:{supporters:{personeel:50,studenten:150}}})

Hier word ook nog een subcategorie gemaakt van eerst supporters met daarin het aantal personeel supporters en het aantal studenten.

## Een record aanpassen waar een aan een voorwaarde voldaan is voor alle records

Volgende commando voegt iets toe waar de naam gelijk is aan “” en je moet multi op true zetten als je op alle records wilt.bv:

## db.films.update({director: 'Erik Van Looy'}, {$set:{genre: "thriller"}}, {multi:true})

## Een collection verwijderen

db.studentenregatta.drop()

## Tonen van de data in je collation

Als de data uit je collation wilt kijken kan je dit doen op twee manieren met find of met pretty extra bv:

db.films.find()

db.films.find()pretty

Dit tweede commando zal de data leesbaardere tonen dan gewoon .find()

## Tonen van bepaalde data

Als je niet alles wilt tonen kan je dit doen door de naam die je wilt gebruiken en deze op 1 te zetten LET OP als je enkel bv de naam wilt moet je de \_id ook wel op 0 zetten wat dit staat standaard altijd er bij. Bv:

db.movies.find({},{title:1,year:1,\_id:0})

## Sorteren

Je kan dit ook sorteren 1 = oplopend of alfabetisch en -1 is dalend of reverse alfabetisch dus bv:

db.movies.find({},{title:1,genres:1}).sort({genre:1})

## Skip en Limit

Dit zijn twee dingen die redelijke eenvoudig zijn. Skip skiped gewoon de zoveel eerste resultaten dus stel je wilt de 5 de dan kan je dit doen door skip van 4 en limit is gewoon laat maar een bepaald aantal resultaten zien dus als je enkel de 5de record wilt kan je dit doen door skip 4 en limit van 1 bv:

db.movies.find().sort({genre:1}).skip(82).limit(15)

## Count

Je kan natuurlijk ook je records die je bekomt gaan tellen met count bv:

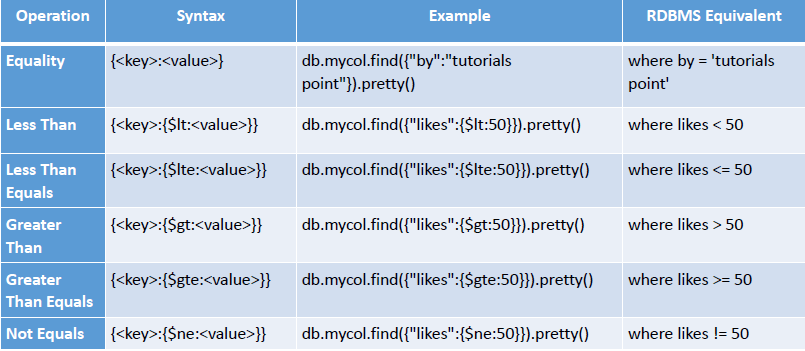
db.movieDetails.find().count()

of bv als je wilt weten hoeveel films er bv gemaakt zijn in een bepaald jaar

db.movies.find({year:1982}).count().pretty()

## Grotere dan, kleiner dan en niet gelijk aan

Je kan ook zoeken op groter dan kleiner dan, groter of gelijk aan, kleiner en gelijk aan en niet gelijk aan.



Bv:

db.movieDetails.find({"runtime":{$gt:120}}).count()

je kan natuurlijk ook werken met letters bij groter dan dus bv de titel begint met een letter tussen L en letter P dus:

db.movies.find( { $and: [ {title: {$gt: "L"} } ,  { title : { $lt : "Q"} }   ] } )

## Opzoeken op naam

Je kan natuurlijk ook zoeken op naam dit doe je door het keyword regex. Daarna zet je het woord dat je zoekt tussen /woord/.

Daarna zijn er nog 3 tekens dat je kan meegeven een ^,$ en i .

^ en $ zet je na de eerste /. ^ wilt zeggen dat het record moet beginnen met dit woord. En $ wilt zeggen dat dit moet eindigen met dit woord. i zet je na de tweede / dit zorgt er voor dat je zoekt op niet hoofdletter gevoelig dus zoekt die op zowel klein als groter letter van dit woord dus bv:

db.movieDetails.find({"actors":{$regex:/^Gary/i}},{title:1,\_id:0}).sort({title:1})

dus moet beginnen met Gary en toon ook als het begint met gary zonder hoofdletter.

Je kan natuurlijk ook perfect werken zonder regex bv:

db.movieDetails.find({'awards.text':/nominations/i},{title:1,'awards.nominations':1,'awards.text':1,\_id:0})

## AND

Als je wilt zoeken dat een waarde gelijk is aan zowel een eerste waarde als ook een tweede waarde kan je dit doen door AND. Dus bv jaar 2000 en actor = naam van actor. Maar je kan ook AND gebruiken als je wilt gaan kijken of het ligt tussen een bepaald jaar tal dan kan je dit doen door bv:

db.movies.find({$and:[{year:{$gte:2014}},{year:{lte:2017}}]},{title:1,year:1,\_id:0}).sort({year:1})

hier ga je de films nemen die liggen tussen 2014 en 2017 inclusief.

## OR

Als je wilt zoeken of het een is of het andere is kan je dit doen moet OR bv:

db.movies.find({$or:[{year:2000},{year:2010}]}).count()

## AND en OR

Je kan natuurlijk ook deze twee combineren in één grote lus. Waar je beide gebruikt.

Er zijn dan twee manieren of een AND met een OR of een OR met een AND.

Dus AND en OR

db.movies.find({title:{$regex:/^West/},$or:[{year:2008},{year:2012}]}).pretty()

Hier moet het dus de naam West zijn EN het jaar 2008 OF 2012.

AND moet niet geschreven worden omdat dit standaard is.

Dus OR en AND

db.movies.find({$or:[{$and:[{year:{$gte:1915}},{year:{$lte:1920}}]},{title:"Western"}]}).pretty()

Hier moet het dus of tussen 1915 en 1920 liggen of met den titel western zijn. Natuurlijk is dit een of dus mag het ook en de titel western omdat dit geen exclusieve of is.

## Subdocumenten

In big data zit je ook met subdocumeten hier kan je ook op gaan zoeken dan met je wel gebruik maken van ‘veldnaam’ als je dit wilt aanroepen bv:

db.movies.find({'awards.wins':20}},{title:1,\_id:0,'awards.nominations':1)

hier is awards een veld en wins zijn subdocument daar van .

zowel bij het zoeken als bij het tonen moet je ‘’ gebruiken.

## Arrays

Je kan zoeken altijd of een waarde in een veld zit dus stel je wilt weten of er in de array een bepaalde naam bevind kan je dit eenvoudig doen door een eenvoudige find.

Als je echt wilt weten op welke plaats dit bepaalde waarde moest staan dan kan je dit doen door een cijfer toe te voegen. Beginnend bij 0 voor de eerste plaats en 1 enzo verder.

Dus bv aantal films waar is opgenomen in Italië

db.movieDetails.find({countries: 'Italy'}).count()

als je wilt dat je enkel het tweede land Italië moet zijn moet je ‘countries.1’ doen dus:

db.movieDetails.find({'countries.1': 'Italy'}).count()

je kan ook bv hier $ne gebruiken dus bv zorg er voor dat het eerste land NIET USA is dus :

db.movieDetails.find({'countries.0':{$ne:'USA'}},{\_id:0,title:1,countries:1}).count()

# Droge Theorie

## Het woord database wordt voor verschillende begrippen gebruikt:

1. de opgeslagen gegevens als zodanig;

2. de wijze waarop de gegevens zijn opgeslagen, zie datamodel;

3. de software waarmee databases kunnen worden aangemaakt en benaderd, zie databasemanagementsysteem (DBMS). Een datawarehouse is een gegevensverzameling die in een zodanige vorm gebracht is dat terugkerende en ad‐hoc vragen in relatief 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. Hierbij wordt dus eigenlijk ‘data’ omgezet naar ’informatie’. Gegevens kunnen in een datawarehouse niet worden ingevoerd of aangepast door gebruikers zelf.

Datamining is het op basis van relevante gegevens gericht zoeken naar (statistische) verbanden in

gegevensverzamelingen met als doel profielen op te stellen voor wetenschappelijk, journalistiek of

commercieel gebruik. Op basis hiervan kunnen dan onderbouwde beslissingen genomen worden. Bv.

is er een verband tussen de leeftijd van de klant en het type shampoo? – is er een verband tussen het

geslacht van de klant en het bedrag waarvoor er maandelijks gekocht wordt in een supermarkt? ‐ ..

Het concept Big Data bestaat al jaren.

Het gaat in hoofdzaak over de evolutie in data en op een andere manier omgaan met data.

## Waarom is Big Data de laatste jaren een hype geworden?

De rijke voedingsbodem voor de Big Data hype bestaat uit:

•Hedendaagse hardwaremogelijkheden, waaronder goedkopere servers;

•Goedkopere en ruimere opslag;

•Mogelijkheden van opensource software;

•Beschikbaarheid van massa’s gegevens.

## Enkele toepassingen van Big Data:

•Voor marketing, zeker bij webbased bedrijven;

•Voor politieonderzoek en ‐opsporing, o.a. bij fraude en cybercrime;

•Voor analyse bij grote datalekken, bijv. WikiLeaks, LuxLeaks, Panama Papers, ...;

•Voor onderzoek in de gezondheidssector, zoals naar ziektes en erfelijkheid;

•Voor de industrie, bijv. om de technologie in auto’s te verbeteren naar veiligheid toe.

## Wanneer spreken we van Big Data?

Er zijn verschillende betekenissen:

• Honderden terabytes;

•Een “klassieke” databank kan de complexiteit niet meer aan. Een alternatief is nodig voor de niet‐relationele gegevens;

•De 3 V’s die aanleiding kunnen geven tot het spreken over Big Data:

O Volume: Een grote hoeveelheid gegevens, teveel om te verwerken op de traditionele manier;

O Velocity: Een hoge snelheid van datatoevoer en een hoge verwerkingssnelheid; O.a. Twitter,

Facebook en andere web toepassingen;

O Variety: De structuur van de gegevens varieert en is niet vast zoals bij een relationele databank.

• In sommige lectuur spreekt men zelfs al over 4 en 5 V’s

O Veracity: betrouwbaarheid van de gegevens

O Value: de info die gegenereerd wordt vanuit de big data moet waardevol zijn bv. verbetert onze gezondheidszorg, veiligheid of optimaliseert een productieproces

•Er is nood aan het opdelen van de data in kleinere eenheden.

Om inzicht te krijgen in Big Data is het noodzakelijk om de architectuur, het gedistribueerde systeem

met verschillende nodes alsook de CAP‐stelling te begrijpen.

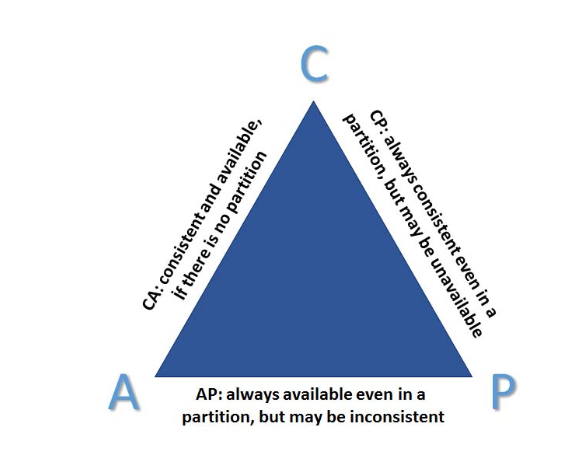
De 3 items nog even verduidelijkt:

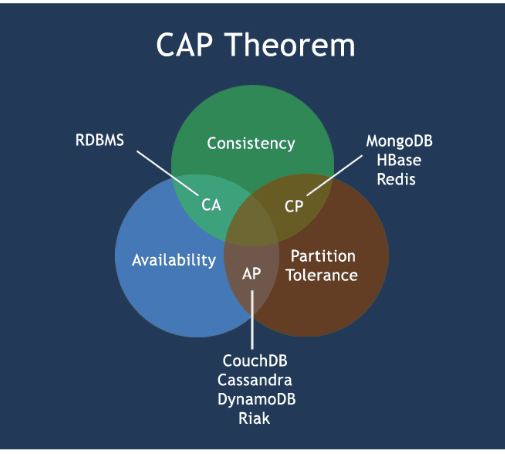
Consistency: Is de mate waarin het DB‐systeem aan alle betrokken servers/nodes de meest recente

gegevens laat zien. De consistency is optimaal als na het uitvoeren van een operatie/transactie (toevoegen, verwijderen of wijzigen van data, acties/taken die gelijktijdig moeten uitgevoerd worden) alle gebruikers dezelfde gegevens zien op alle betrokken servers/nodes.

Availability: Is de mate waarin het DB‐systeem beschikbaar is. De availability is optimaal wanneer het systeem altijd beschikbaar, online is. En de gebruiker altijd een respons krijgt. Dit kenmerk is van groot belang bij web toepassingen en datawarehouseapplicaties. Bv. het is zeer belangrijk dat je onmiddellijk je facebookapplicatie kan zien, maar het kan zijn dat je nog altijd een “facebookvriend”

kan bereiken alhoewel hij je net “gedefriend” heeft of dat je bepaalde berichten iets later krijgt dan andere facebookgebruikers. Partition tolerance: Is de mate waarin het DB‐systeem blijft werken bij uitval van 1 of meerdere server(s)/node(s). Het is belangrijk dat je computersysteem blijft functioneren ook als één of meerdere nodes zouden uitvallen (als de communicatie tussen servers tijdelijk niet betrouwbaar is). De servers moeten niet constant met elkaar kunnen communiceren. Deze tolerance is vooral belangrijk als er een groot volume aan data is, verdeeld over verschillende servers. De CAP‐stelling zegt dat als het gaat om een gedistribueerd systeem met meerdere servers/nodes er altijd een keuze moet gemaakt worden tussen consistency en availability. Eén van beide garanties kan dan niet vervuld worden. Bij het kiezen van een databasesysteem is het dus zeer belangrijk vooraf na te gaan waar de prioriteiten liggen. Waar situeren de noden zich in onderstaande driehoek?





Bij relationele databanken zijn Consistency en Availability (CA) het belangrijkst, omdat bij dit type

databanken hoofdzakelijk gewerkt wordt met operationele gegevens die constant consistent en

bereikbaar moeten zijn. Bv. MySQL, MariaDB, Oracle SQL, Postgre SQL, Microsoft SQL Server.

Bij NoSQL databanken zijn 2 combinaties mogelijk:

•CP:sommige data is niet direct bereikbaar, maar de data is wel consistent (dezelfde voor elke server/node) en de data mag gespreid zijn over meerdere servers;

•AP: het systeem is beschikbaar maar niet altijd volledig. Dit is vooral nuttig bij analyse van

Big Data waarbij de consistency minder belangrijk is en de data verdeeld is over verschillende nodes. Je kan dit vergelijken met het domeinnamensysteem (DNS). Niet iedereen zal gelijktijdig wijzigingen zien maar wel na een paar dagen. Je moet daar niet op wachten om verder te kunnen werken.

Er zijn 2 belangrijke database principes, nl. ACID en BASE. Op elk van deze principes gaan we dieper in.

## ACID

Een klassieke relationele databank volgt het principe van ACID. Dit principe stelt dat een transactie (nieuw gegeven toevoegen, bestaand gegeven wijziging of verwijderen, acties/taken die gelijktijdig moeten uitgevoerd worden) in de database Atomic,Consistent, Isolated en Durable moet zijn.

Atomic

Elke transactie slaagt volledig of slaagt niet. Als 1 statement een fout geeft dan is er een rollback voor

de hele transactie. Bv. als we producten verkopen, dan moeten gelijktijdig de producten gereserveerd worden en de voorraad aangepast worden. In een relationele databankomgeving zal

een operatie niet slagen als niet elke deelactie kan uitgevoerd worden.

Consistent

Is niet hetzelfde als Consistency bij CAP !!!!!!!!

Gegevens mogen niet tegenstrijdig worden. Er moet op elk moment aan alle constraints voldaan zijn.

Dus als een bepaald statement uit een transactie de gegevens tegenstrijdig zou maken dan wordt de

transactie automatisch afgebroken. Referentiële integriteit van de database waakt hierover (zie Foreign Keys bij normalisatie, ERD). Deze vereiste is zeer belangrijk bij operationele gegevens in bedrijven o.a. stockgegevens, gegevens, boekhoudkundige gegevens. Dit is minder belangrijk in

een datawarehouse‐omgeving, waarin de transactiesnelheid zeer belangrijk is als gegevens worden opgevraagd en/of geanalyseerd (bv. om rapporten en/of grafieken te maken).

Isolated

Elke transactie wordt los van een andere transactie uitgevoerd; dit wil zeggen dat transacties die

gelijktijdig worden uitgevoerd elkaars tussenresultaten niet kunnen zien.

Durable

Als de transactie voltooid is (bevestigd met een impliciete of expliciete commit), dan is dit permanent/onomkeerbaar.

## BASE

Het ACID‐principe is niet houdbaar voor alle hedendaagse toepassingen. Bijgevolg wordt er een

ander principe toegepast in NoSQL‐databanken. Daar spreekt men over een BASE‐systeem

waarin het consistent zijn van de databank voor een deel wordt opgegeven.

Basic Availability

Het systeem garandeert availability zoals voorzien in de CAP‐stelling. Er mogen tijdelijk

onvolkomenheden zitten in de opgeslagen data of er kunnen tijdelijk onbereikbare data zijn. Het gaat hier in elk geval toch maar om een zeer beperkt aantal gegevens.

Soft state

Het systeem kan in de tijd wijzigen, zelfs zonder input van data. Het is de verantwoordelijkheid van de ontwikkelaar om die tijdelijke niet‐consistentie op te vangen.

Eventual consistency

Op den duur zal de databank consistent worden. Bv. een wijziging zal niet onmiddellijk voor iedereen zichtbaar zijn, maar wel na verloop van een paar dagen (web toepassingen).

Opgelet: de Engelse term ‘eventual’ betekent uiteindelijk, niet misschien!

## Distributed datastores

Een distributed computersysteem bestaat uit verschillende computers en software componenten die

via een netwerk communiceren

Grote voordelen van een dergelijk systeem zijn:

•Reliability: als één of meer nodes crashen dan kunnen de overige blijven verder werken;

•Scalability: uitbreiden is altijd mogelijk;

•Sharing resources: vele applicaties maken gebruik van dezelfde gegevens en systemen

(bv. dure printers);

•Flexibility: eenvoudig om nieuwe services te installeren, te implementeren en te debuggen

•Speed: meer power en snelheid;

•Open system: elke client (gebruiker) kan beschikken over elke service indien voldoende

rechten;

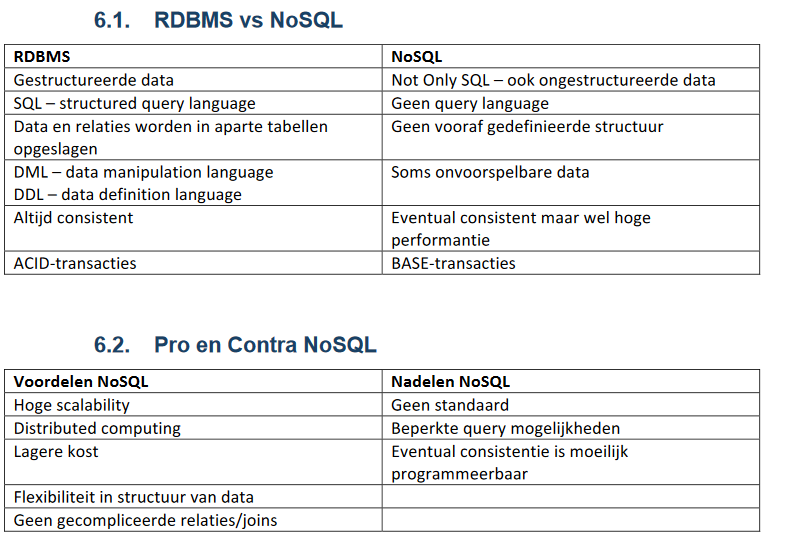
•Performance: hogere performantie is mogelijk doordat de taken verdeeld kunnen worden

over verschillende nodes.

Nadelen zijn: minder software support, netwerkproblemen, beveiligingsproblemen.

## NoSQL

Een NoSQL DBMS is een niet‐relationeel databasemanagement systeem speciaal ontwikkeld voor distributed data stores die big data moeten bevatten. Dit type dataopslag vereist geen vaste structuren en vermijdt join‐operaties waarbij gegevens van verschillende tabellen moeten gekoppeld worden aan elkaar.



## Enkele eigenschappen van key‐value stores:

•Meest gebruikte basistype;

•Kan vele TB aan gegevens aan;

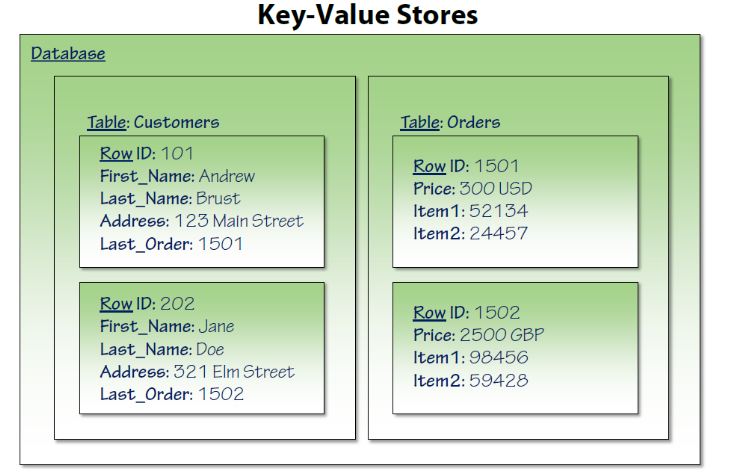
•Laten ongestructureerde gegevens toe, gemakkelijk uitbreidbaar;

•Gegevens worden opgeslagen als een hash table waarin elke key uniek is en de value een string, JSON‐object, BLOB‐object, ... kan zijn;

•Een key‐value pair kan bestaan uit een naam gecombineerd met een waarde;

•Bv.: Dynamo, Redis (in memory database d.w.z. gegevens staan inhet RAM‐geheugen, geen indexen), Oracle NSQL Database;

•beperking: je kan enkel zoeken via de key!



## Enkele eigenschappen van column‐oriented stores:

•Werken met de kolommen en elke kolom wordt apart behandeld en kan deel uitmaken van een column family (CF);

•Slaan de values van een kolom aaneengesloten op;

•Slaan de kolomgegevens op in specifieke files;

•Gebruiken keys, maar die verwijzen naar verschillende kolommen;

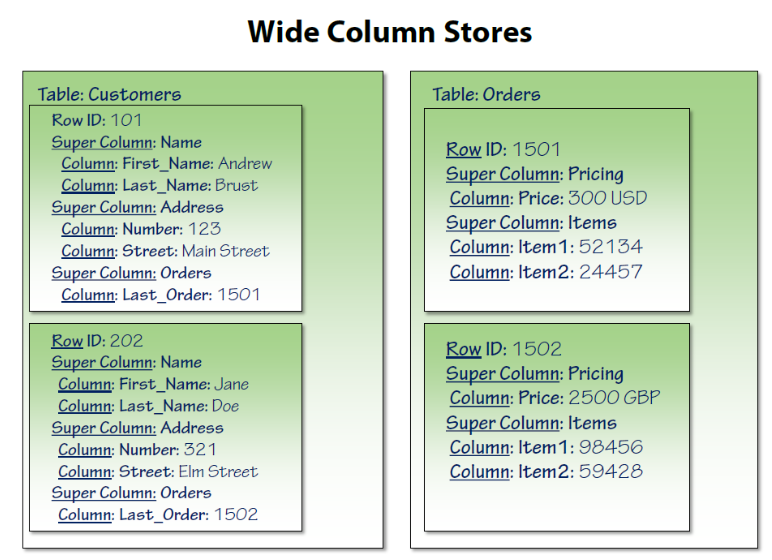
•Laten ook queries toe;

•Alle data in een kolomfile zijn van hetzelfde type, waardoor zegemakkelijk gecompressed

kunnen worden;

•Hebben daardoor ook een hoge performance bij gewone en groepsqueries en zijn dan ook zeer geschikt voor Business Intelligence(BI) en Customer Relationship Management (CRM) toepassingen;

•Bv. HBase, Cassandra, SimpleDB, SAP HANA.



## Enkele eigenschappen van document‐oriented stores:

•Bevatten een collection van documenten;

•slaan de data op in documenten waarbij de key toegang geeft tot de waarden (key/value pair);

•hebben niet noodzakelijk een vaste structuur, waardoor ze flexibel en gemakkelijk aanpasbaar zijn;

• slaan documents op in collections om diverse data te groeperen en deze documents kunnen

verschillende key‐value pairs en zelfs geneste documenten bevatten;

•de documenten zijn JSON objecten;

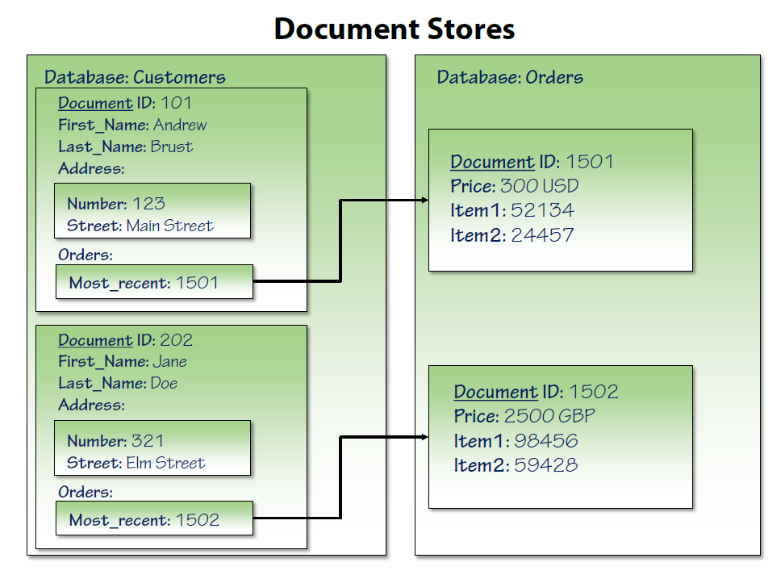
•eigenlijk een gespecialiseerde key‐value store met specifieke eigenschappen maar zonder de

beperking van KeyValue nl. je kan nu wel zoeken via andere indexen;

•vanuit applicaties kan er naar de documenten verwezen worden via URI’s

•biedt een API of query language zodat er kan gezocht worden op de inhoud van het document;

•Bv. MongoDB, CouchDB, Couchbase.



## Enkele eigenschappen van graph stores:

•Slaan data op in een grafiek;

•Presenteren gegevens op een zeer toegankelijke manier;

•Zijn een verzameling van nodes en edges;

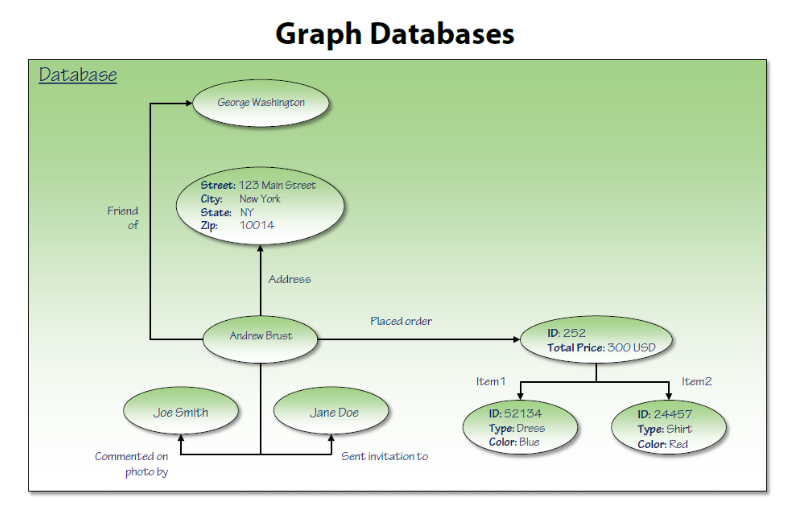
•Elke node vertegenwoordigt een entiteit (student, klant,...) en elke edge vertegenwoordigt een connectie/relatie tussen 2 nodes;

•Elke node en elke edge hebben een unieke identiteit;

•Elke node kent zijn aangrenzende nodes;

•Gebruikt indexen voor opzoekingen;

•Bv. OrientDB, Neo4J, Apache Giraph



In een bedrijf, organisatie zal er meestal gebruik gemaakt worden van verschillende types datastores omdat de gebruikte gegevens zeer uiteenlopend kunnen zijn en dus een aangepast systeem vereisen.