Basisprincipes Big Data en NoSQL

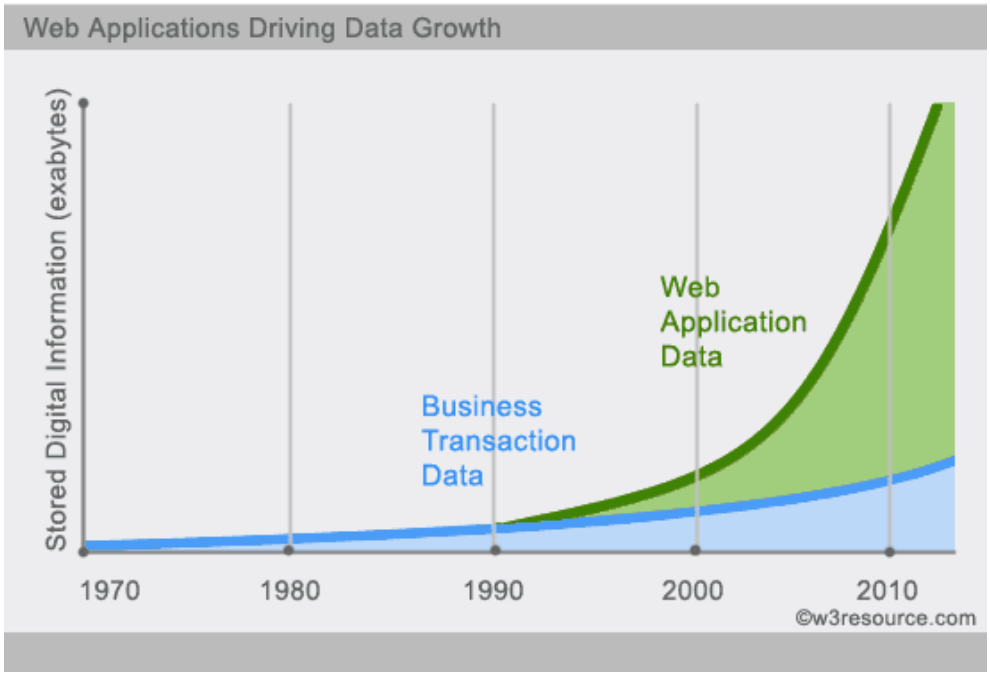
Isabelle Godfrind
Carina Medats
Kerstin Nys

Inhoud

1. Inleiding

Naast relationele databanken (standaard om gestructureerde gegevens op te slaan) bestaan er nog andere types.

Met de boom rond social media beschikken sommige grote bedrijven over massa's interessante gegevens o.a. tweets, facebook, weblogs, …. maar ook allerhande feeds, RFID-scans[1], sensordata[2], clickstreamdata[3] die vaak anders gestructureerd zijn en niet opgeslagen worden in een relationele databank.



Figuur 1: Datagroei - bron: http://www.w3resource.com/mongodb/nosql.php

Daarvoor zijn nieuwe infrastructuur, nieuwe programmeeromgevingen en nieuwe data-omgevingen nodig.

"Om de waarde van uw gegevens ten volle te kunnen benutten, hebt u een volledig platform nodig dat zowel gestructureerde als ongestructureerde gegevens met veiligheid, betrouwbaarheid en consistentie kan beheren. Datawarehouse-oplossingen en Big Data-oplossingen […] bieden een vertrouwde infrastructuur die elk gegevenstype kan verwerken en van terabytes naar petabytes kan schalen met realtime-prestaties."[4]

---

[1] RFID is een afkorting van Radio Frequency IDentification, identificatie met radiogolven. RFID is een methode om van een afstand informatie op te slaan en te lezen van zogenaamde RFID `tags` die op of in objecten zitten. Deze tags kunnen `actief` of `passief` zijn. RFID wordt o.a. gebruikt bij magazijnbeheer van pakketten goederen, waarbij het pakket voorzien wordt van een RFID-tag.

[2] Sensordata is data afkomstig van apparaten die voorzien zijn van 1 of meer sensoren, o.a. meetapparatuur voor lucht, water, gassen, grondsamenstelling, maar ook smartphones, smartwatches, …

[3] "A clickstream is the recording of the parts of the screen a computer user clicks on while web browsing or using another software application. As the user clicks anywhere in the webpage or application, the action is logged on a client or inside the web server, as well as possibly the web browser, router, proxy server or ad server. Clickstream analysis is useful for web activity analysis, software testing, market research and for analyzing employee productivity." - https://en.wikipedia.org/wiki/Clickstream

[4] https://www.microsoft.com/nl-be/server-cloud/data-management/

2. Terminologie

Een verklaring voor enkele belangrijke datatermen is hier niet misplaatst om de rest van dit hoofdstuk te begrijpen.

Een **database**, gegevensbank of databank is een digitaal opgeslagen archief, ingericht met het oog op flexibele dataopslag, raadpleging en gebruik. Databases spelen een belangrijke rol voor het archiveren en actueel houden van gegevens bij onder meer de overheid, financiële instellingen en bedrijven, in de wetenschap, en worden op kleinere schaal ook privé gebruikt.

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.
Om losse data vanuit het bronsysteem om te zetten in informatie is het noodzakelijk dat de gebeurtenis of voorwerp waarover de data handelt, bekend is. Slechts dan krijgen de losse waarden een betekenis. Wil data betekenis hebben, dan is kennis van het kader waarbinnen ze verzameld zijn noodzakelijk.[5]

<u>Voorbeeld:</u>
Een onderhoudsbedrijf beheert en controleert de cv-ketels van een groot aantal mensen. Op een zeker moment komt er op het bewakingsscherm de tekst [foutcode 2] te staan. Als niet bekend is welk systeem deze foutcode genereert en wat dat systeem met deze foutcode bedoelt, dan is dit een waarde zonder verdere betekenis. Pas als deze foutcode vertaald wordt naar zijn betekenis, dus de waarde wordt omgezet naar informatie, dan krijgt het betekenis en dan is het beschikbaar voor verdere verwerking. Zo kan de foutcode bij een cv-ketel van merk A staan voor *gasdruk te laag*, maar bij merk B juist voor *waterniveau te laag*.

Bij het opslaan van deze data in een informatiesysteem is het noodzakelijk om de data om te zetten in informatie. In dit geval volstaat het niet om alleen *foutcode 2* op te slaan, maar moet deze foutcode ook vertaald worden naar een begrijpelijke foutmelding. Dit laatste is belangrijk indien er onderzoek wordt gedaan naar de storingen in cv-ketels. Het is niet van belang om te weten wanneer een ketel *foutcode 2* (=losse data) vermeldt, omdat de betekenis van deze code afhankelijk is van het type cv-ketel. Er kan wel gezocht worden op de foutmelding *waterniveau te laag* (=informatie).

[5] Succes met Big Data, Wiebe van de Zee & Bert van der Zee, Van Duuren Media

Informatie wordt nuttig wanneer deze verzameld wordt over een langere periode. Door veranderingen in informatie te analyseren, kan men patronen ontdekken. Op basis van deze patronen kunnen dan de nodige acties ondernomen worden.

Terug naar ons voorbeeld:
Bij de analyse van de foutmeldingen van de bewaakte cv-ketels valt op dat de melding *waterniveau te laag* over een langere periode het meest voorkomt. Nadere analyse toont aan dat deze melding piekt in de maand september, het moment dat de meeste huishoudens de cv-ketel na een zomer stilstand weer inschakelen. Naar aanleiding van deze analyse maakt het onderhoudsbedrijf een instructie voor haar klanten over het opnieuw starten van de cv-ketel na de zomerstop, in de hoop dat daarmee het aantal meldingen vermindert.

In relationele databanken worden meestal de normalisaties uitgevoerd vooraleer de gegevens naar de databank worden overgezet. Bv. berekende velden worden geweerd uit een relationele database opdat de berekeningen telkens opnieuw moeten gemaakt worden met de “recentste” gegevens. Dit is echter niet meer nodig in een datawarehouse-omgeving omdat daarin de gegevens meestal enkel nog gebruikt worden voor BI-doeleinden (zie verder). Dus daar is het belangrijker dat de opvragingen snel kunnen gebeuren en daarom kunnen daarin best wel de resultaten van bepaalde berekeningen opgenomen worden.

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? - ….

3. Big Data

3.1. Het concept Big Data bestaat al jaren.

In de jaren 50 van de vorige eeuw gebruikten bedrijven basis analyses om inzichten en trends te onderzoeken. Ze stopten gegevens in tabellen en gingen met behulp van wiskunde en/of statistiek manueel berekeningen maken, maar het aantal gegevens was meestal beperkt. Vaak was er geen tijd om de verzamelde gegevens te analyseren. Later werden daarvoor applicaties gebruikt zoals spreadsheets en databasetoepassingen. Met zulke analyses wilden/willen bedrijven beslissingen voor de toekomst nemen. We spreken dan van BI-oplossingen (Business Intelligence).

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

3.2. 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.

3.3. 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:
    - Volume: Een grote hoeveelheid gegevens, teveel om te verwerken op de traditionele manier;
    - Velocity: Een hoge snelheid van datatoevoer en een hoge verwerkingssnelheid; O.a. Twitter, Facebook en andere webtoepassingen;
    - 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
    - Veracity: betrouwbaarheid van de gegevens
    - 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.

3.4. Hoe werkt Big Data?

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. Hiervoor reiken we een aantal video's aan uit Pluralsight.

3.4.1. Architectuur

Bekijk deze Pluralsight video (3'10"): https://app.pluralsight.com/player?author=ben-sullins&name=data-analytics-hands-on-m9&mode=live&clip=3&course=data-analytics-hands-on

3.4.2. Distributed Systems

Bij Big Data gaat het niet alleen om grote hoeveelheden data, maar ontbreekt ook elke structuur in de data. Aangezien de verwerkingstijd schaalt met de hoeveelheid informatie heeft dit tot gevolg dat een andere aanpak nodig is.

Indien een bepaalde server te lang nodig heeft om individuele opdrachten uit te voeren, dan zijn er 3 mogelijkheden om de verwerking te versnellen:

- Snellere server: hierbij is één computer in staat om de informatie sneller te verwerken. In deze optie is het bijvoorbeeld moeilijk om de verwerkingstijd met dertig procent terug te brengen ten koste van een verdubbelde investering. Bovendien is deze aanpak gebonden aan de maximale verwerkingssnelheid van de moderne computer en die is niet oneindig groot. Ook betekent dit dat de computer vervangen moet worden en dat alle programma's en informatie moet worden overgezet. Dit laatste vraagt (veel) tijd en gedurende de vervangingsperiode is er geen mogelijkheid tot verwerken van informatie.
- Meer servers: dit wil zeggen dat het werk gedaan wordt door verscheidene computers. In deze situatie betekent een dubbele investering meteen een dubbele verwerkingssnelheid. Worden direct tien computers geïnstalleerd, dan zou de verwerkingssnelheid theoretisch vertienvoudigen. Mocht de versnelling nog niet voldoende zijn, dan is het bijplaatsen van meer computers een relatief eenvoudige actie.
- Optimaliseren van gebruikte programma's: deze optie staat meestal alleen open voor ontwikkelaars en programmeurs. In hoeverre deze oplossing haalbaar is hangt af van de kosten voor het optimaliseren en de bereikte winst in verwerkingssnelheid. Overigens is deze aanpak altijd goed (mits economisch verantwoord) omdat het altijd winst oplevert.

In de praktijk wordt het gedistribueerd werken (meer servers) gecombineerd met geoptimaliseerde programma's om zodoende tot een hoge verwerkingssnelheid te komen. Deze combinatie geeft de beste resultaten en ook de eenvoudigste en goedkoopste weg naar het vergroten van de capaciteit. Dit leidt dan tot een gedistribueerd computersysteem.

Een gedistribueerd computersysteem bestaat uit verschillende computers en softwarecomponenten die via een netwerk (lokaal LAN of worldwide WAN) communiceren. Het systeem kan bestaan uit verschillende mainframes, workstations en PC's. De servers worden ook nodes genoemd. De computers communiceren met elkaar en delen de systeembronnen om een gemeenschappelijk doel te bereiken. De data kan gespreid zijn over verschillende nodes en de taken worden meestal verdeeld over verschillende nodes.

In de volgende video wordt hierover een beknopte uitleg gegeven (2'23"):
https://app.pluralsight.com/player?author=ben-sullins&name=data-analytics-hands-on-m9&mode=live&clip=4&course=data-analytics-hands-on

Distributed datastores zijn datastores die inzetbaar zijn in een dergelijk distributed systeem. Daarvoor komen geen RDBMS in aanmerking en hebben we dus NoSQL-datastores nodig.

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.

3.4.3. CAP-stelling

Als je gebruik maakt van databanken en meer bepaald van Distributed File Systems dan is de **CAP**-stelling heel belangrijk. Het CAP-theorema werd in 1995 voor het eerst gepresenteerd door Eric Brewer en wordt daarm ook wel *Brewer's theorema* genoemd. Dit theorema beschrijft de gevolgen van het distribueren van een database over verschillende (via een netwerk verbonden) computers. De letters CAP staan voor *consistency, availability* en *partition tolerance*.

In de volgende video wordt hierover uitleg gegeven (4'55''):

https://app.pluralsight.com/player?author=ben-sullins&name=data-analytics-hands-on-m9&mode=live&clip=5&course=data-analytics-hands-on

Nog een interessante link: http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

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 webtoepassingen 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.

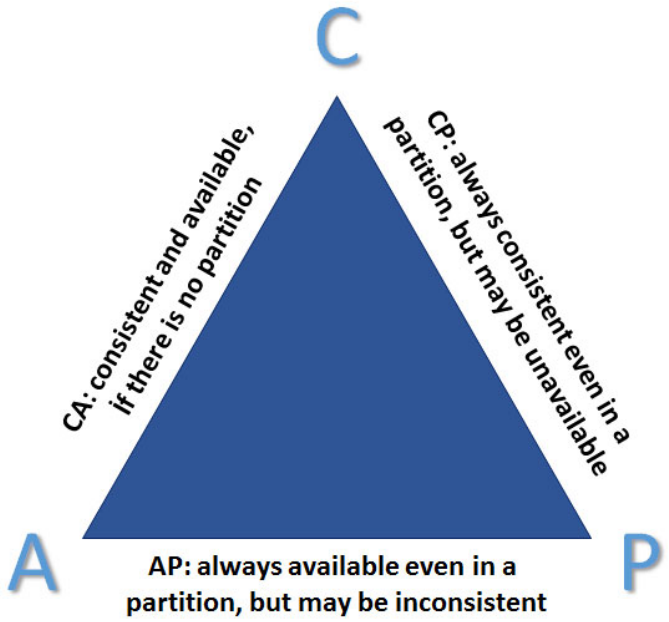
Partition tolerance:

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.

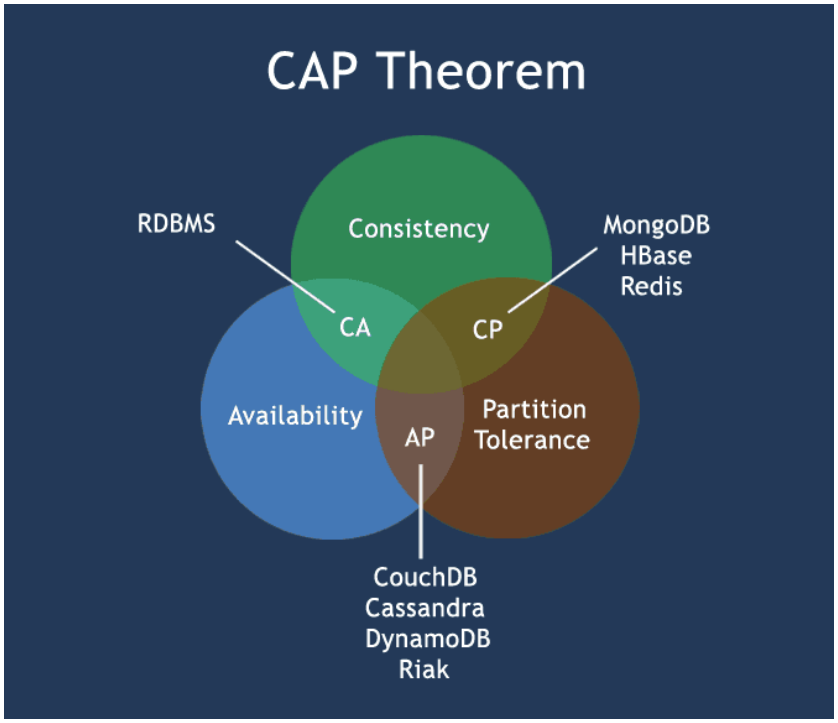
De CAP-stelling zegt dat als het gaat om een **gedistribueerd systeem** met meerdere servers/nodes er altijd een keuze moet gemaakt worden tussen consistency en availability. Eén van beide garanties kan dan niet vervuld worden.

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?



Figuur 2 Waar bevinden de noden van het bedrijf zich?

Waar het bedrijf zich bevindt qua noden zal mede bepalend zijn bij het maken van een keuze in het ruime aanbod van DBMS-systemen.



Figuur 3: CAP-stelling

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, PostgreSQL, 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.

Wat we zeker niet uit het oog mogen verliezen is dat er ook een keerzijde aan dit Big Dataverhaal zit, nl. privacy. Willen we dat iedereen die dat wil, weet waar we hebben ingecheckt, wat we doen, welke producten we kopen, in welke winkel we iets gekocht hebben, … ?

4. Database principes

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

4.1. 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.

Alvorens deze begrippen uit te leggen, wordt toegelicht wat een transactie is.

4.1.1. Transactie

"In a database system a transaction might consist of one or more data-manipulation statements and queries, each reading and/or writing information in the database. Users of database systems consider consistency and integrity of data as highly important. A simple transaction is usually issued to the database system in a language like SQL wrapped in a transaction, using a pattern similar to the following:

1. Begin the transaction
2. Execute a set of data manipulations and/or queries
3. If no errors occur then commit the transaction and end it
4. If errors occur then roll back the transaction and end it

If no errors occurred during the execution of the transaction then the system commits the transaction. A transaction commit operation applies all data manipulations within the scope of the transaction and persists the results to the database. If an error occurs during the transaction, or if the user specifies a rollback operation, the data manipulations within the transaction are not persisted to the database. In no case can a partial transaction be committed to the database since that would leave the database in an inconsistent state."[6]

4.1.1. 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.

4.1.2. 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, financiële

[6] https://en.wikipedia.org/wiki/Database_transaction

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).

4.1.3. Isolated

Elke transactie wordt los van een andere transactie uitgevoerd; dit wil zeggen dat transacties die gelijktijdig worden uitgevoerd elkaars tussenresultaten niet kunnen zien.

4.1.4. Durable

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

4.2. 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.

4.2.1. 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.

4.2.2. 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.

4.2.3. 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 (webtoepassingen).
Opgelet: de Engelse term ‘eventual’ betekent uiteindelijk, niet misschien!

“Instead, NoSQL relies upon a softer model known, appropriately, as the BASE model. This model accommodates the flexibility offered by NoSQL and similar approaches to the management and curation of unstructured data. BASE consists of three principles:

- **Basic Availability**. The NoSQL database approach focuses on the availability of data even in the presence of multiple failures. It achieves this by using a highly distributed approach to database management. Instead of maintaining a single large data store and focusing on the fault tolerance of that store, NoSQL databases spread data across many storage systems with a high degree of replication. In the unlikely event that a failure disrupts access to a segment of data, this does not necessarily result in a complete database outage.

- **Soft State**. BASE databases abandon the consistency requirements of the ACID model pretty much completely. One of the basic concepts behind BASE is that data consistency is the developer's problem and should not be handled by the database.
- **Eventual Consistency**. The only requirement that NoSQL databases have regarding consistency is to require that at some point in the future, data will converge to a consistent state. No guarantees are made, however, about when this will occur. That is a complete departure from the immediate consistency requirement of ACID that prohibits a transaction from executing until the prior transaction has completed and the database has converged to a consistent state."

5. 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.

5.1. RDBMS vs NoSQL

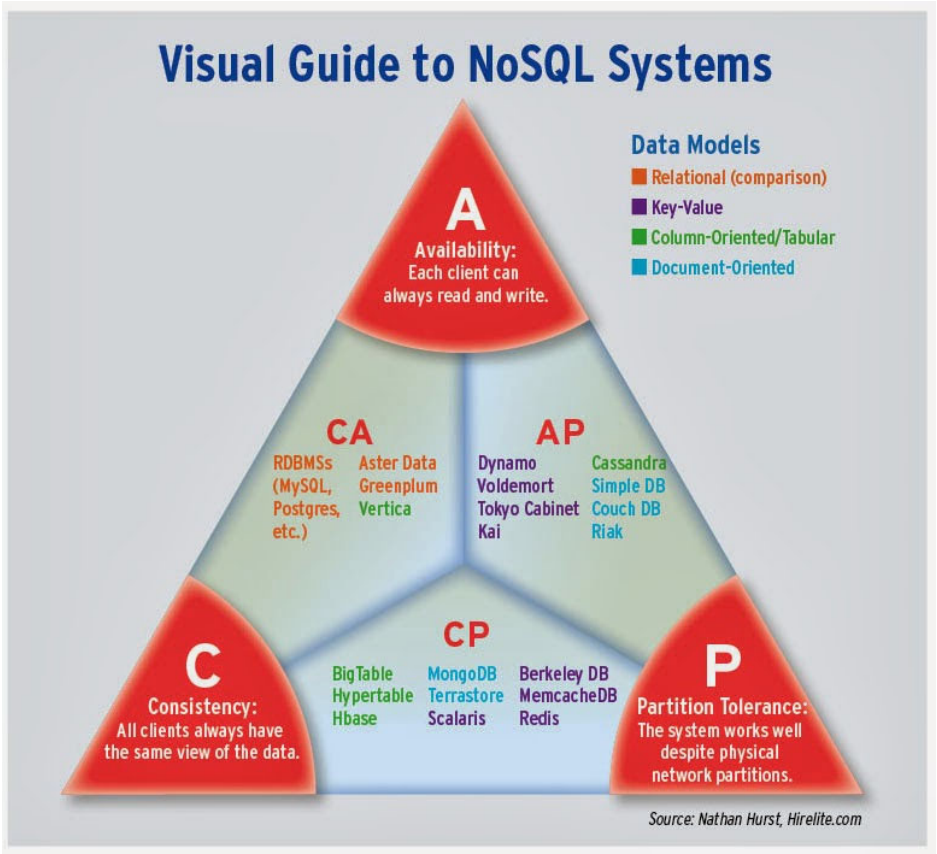
RDBMS	**NoSQL**
Gestructureerde data	Not Only SQL – ook ongestructureerde data
SQL – structured query language	Geen standard query language
Data en relaties worden in aparte tabellen opgeslagen	Geen vooraf gedefinieerde structuur
DML – data manipulation language DDL – data definition language	Soms onvoorspelbare data
Altijd consistent	Eventual consistent maar wel hoge performantie
ACID-transacties	BASE-transacties

5.2. Pro en Contra NoSQL

Voordelen NoSQL	**Nadelen NoSQL**
Hoge scalability	Geen standaard
Distributed computing	Beperkte query mogelijkheden
Lagere kost	Eventual consistentie is moeilijk programmeerbaar
Flexibiliteit in structuur van data	
Geen gecompliceerde relaties/joins	

5.3. NoSQL database systemen

In onderstaand figuur krijg je een summier overzicht van de belangrijkste databasesystemen die er momenteel op de markt zijn. Ze zijn ingedeeld volgens de principes van de CAP-stelling en de kleur geeft aan om welk type database het gaat.

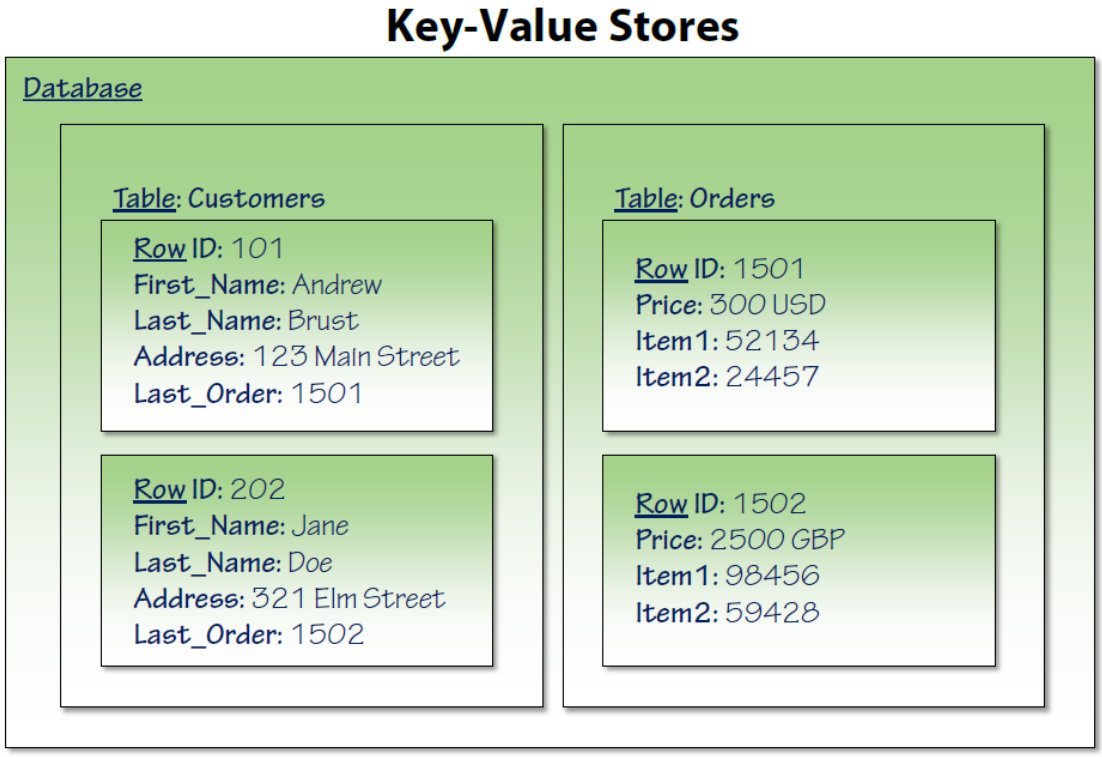


Figuur 4 CAP theorem met DBMS en hun type

Hieronder lichten we de volgende data modellen toe: key-value stores, column-oriented stores, document-oriented stores en graph stores.

5.3.1. Key-value stores

Zie ook Pluralsight: https://app.pluralsight.com/player?course=understanding-nosql&author=andrew-brust&name=understanding-nosql-m1-tech-breakdown&clip=0&mode=live&start=313.083998&noteid=b4515986-a34e-4593-a9e1-ce2874bf44b9



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 in het RAM-geheugen, geen indexen), Oracle NSQL Database;
- beperking: je kan enkel zoeken via de key!

Bij noSQL database systemen zijn er geen foreign keys die de tabellen verbinden. In bovenstaand voorbeeld is er dus geen verbinding tussen Last_Order van de tabel Customer en de Row ID van de tabel Orders.

Alle info die gezamenlijk nodig is, staan samen in tabellen. Deze tabellen zijn vaak flexibel en uitbreidbaar.

5.3.2. Column-oriented stores

We zullen eerst een tabel uit een relationeel database management systeem bespreken, om daarna over te gaan naar een kolom georiënteerd systeem. Vervolgens zal het verschil tussen beide systemen worden aangehaald.

A relational database management system provides data that represents a two-dimensional table, of columns and rows. For example, a database might have this table:

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	40000
002	12	Jones	Mary	50000
003	11	Johnson	Cathy	44000
004	22	Jones	Bob	55000

This simple table includes an employee identifier (EmpId), name fields (Lastname and Firstname) and a salary (Salary). This two-dimensional format is an abstraction. In an actual implementation, storage hardware requires the data to be serialized into one form or another.

A common method of storing a table is to serialize each row of data, like this;

```
001:10,Smith,Joe,40000;
002:12,Jones,Mary,50000;
003:11,Johnson,Cathy,44000;
004:22,Jones,Bob,55000;
```

As data is inserted into the table, it is assigned an internal ID, the `rowid` that is used internally in the system to refer to data. In this case the records have sequential rowids independent of the user-assigned empid. In this example, the DBMS uses short integers to store rowids. In practice, larger numbers, 64-bit or 128-bit, are normally used.

Row-based systems are designed to efficiently return data for an entire row, or record, in as few operations as possible.

Row-based systems are not efficient at performing set-wide operations on the whole table, as opposed to a small number of specific records. For instance, in order to find all records in the example table with salaries between 40.000 and 50.000, the DBMS would have to fully scan through the entire table looking for matching records. While the example table shown above will likely fit in a single disk block, a table with even a few hundred rows would not, and multiple disk operations would be needed to retrieve the data and examine it.

To improve the performance of these sorts of operations (which are very common, and generally the point of using a DBMS), most DBMSs support the use of database indexes, which store all the values from a set of columns along with rowid pointers back into the original table. An index on the salary column would look something like this:

```
001:40000;
003:44000;
002:50000;
004:55000;
```

As they store only single pieces of data, rather than entire rows, indexes are generally much smaller than the main table stores. Scanning this smaller set of data reduces the number of disk operations. If the index is heavily used, it can dramatically reduce the time for common operations. However, maintaining indexes adds overhead to the system, especially when new data is written to the database. Records not only need to be stored in the main table, but any attached indexes have to be updated as well.

A number of row-oriented databases are designed to fit entirely in RAM, an in-memory database. These systems do not depend on disk operations, and have equal-time access to the entire dataset. This reduces the need for indexes, as it requires the same amount of operations to full scan the original data as a complete index for typical aggregation purposes. Such systems may be therefore simpler and smaller, but can only manage databases that will fit in memory.

A column-oriented database serializes all of the values of a column together, then the values of the next column, and so on. For our example table, the data would be stored in this fashion:

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
40000:001,50000:002,44000:003,55000:004;
```

In this layout, any one of the columns more closely matches the structure of an index in a row-based system. This may cause confusion that can lead to the mistaken belief a column-oriented store "is really just" a row-store with an index on every column. However, it is the mapping of the data that differs dramatically. In a row-oriented indexed system, the primary key is the rowid that is mapped from indexed data. In the column-oriented system, the primary key is the data, which is mapped from rowids.[3] This may seem subtle, but the difference can be seen in this common modification to the same store:

```
…;Smith:001;Jones:002,004;Johnson:003;…
```

As two of the records store the same value, "Jones", it is possible to store this only once in the column store, along with pointers to all of the rows that match it. For many common searches, like "find all the people with the last name Jones", the answer is retrieved in a single operation. Other operations, like counting the number of matching records or performing math over a set of data, can be greatly improved through this organization.

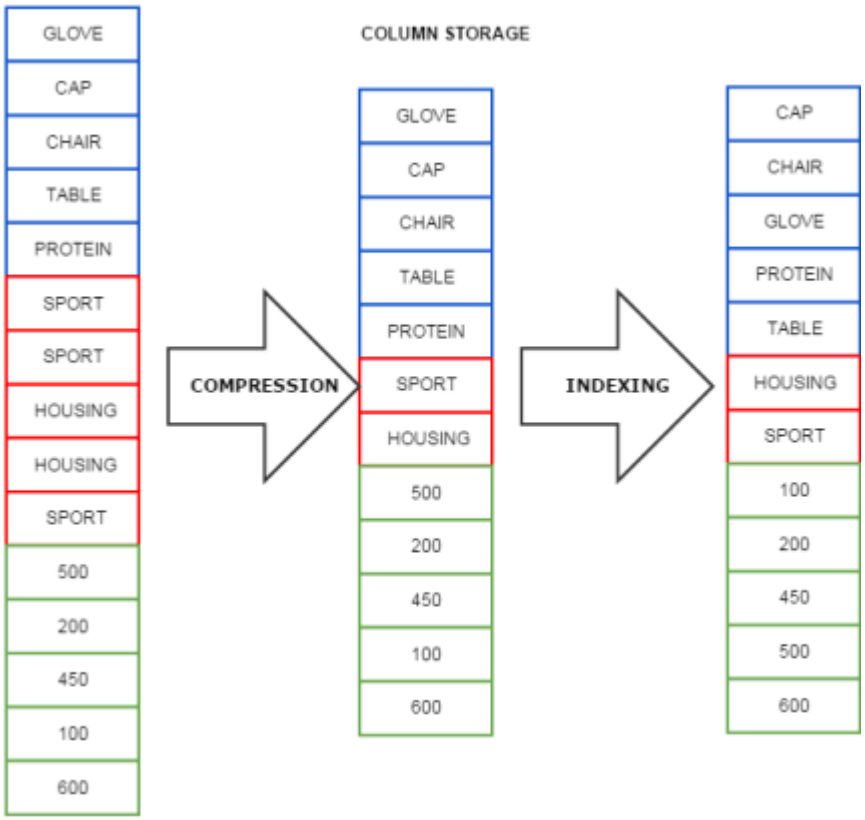
2de Voorbeeld[7]

LOGICAL TABLE STRUCTURE

MATERIAL	CATEGORY	REVENUE (EUR)
GLOVE	SPORT	500
CAP	SPORT	200
CHAIR	HOUSING	450
TABLE	HOUSING	100
PROTEIN	SPORT	600

ROW STORAGE

[7] http://teachmehana.com/row-store-vs-column-store/



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 ze gemakkelijk 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.

Zie Pluralsight: https://app.pluralsight.com/player?course=understanding-nosql&author=andrew-brust&name=understanding-nosql-m1-tech-breakdown&clip=3&mode=live&start=191.175816&noteid=5757f4b9-450a-470e-87d2-8a6960d41904

5.3.3. Document-oriented stores

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 [8];
- biedt een API of query language zodat er kan gezocht worden op de inhoud van het document;
- Bv. MongoDB, CouchDB, Couchbase.

[8] Uniform Resource Identifier = formele beschrijving om een bron te kunnen benaderen vergelijkbaar met URL (Uniform Resource Locator)

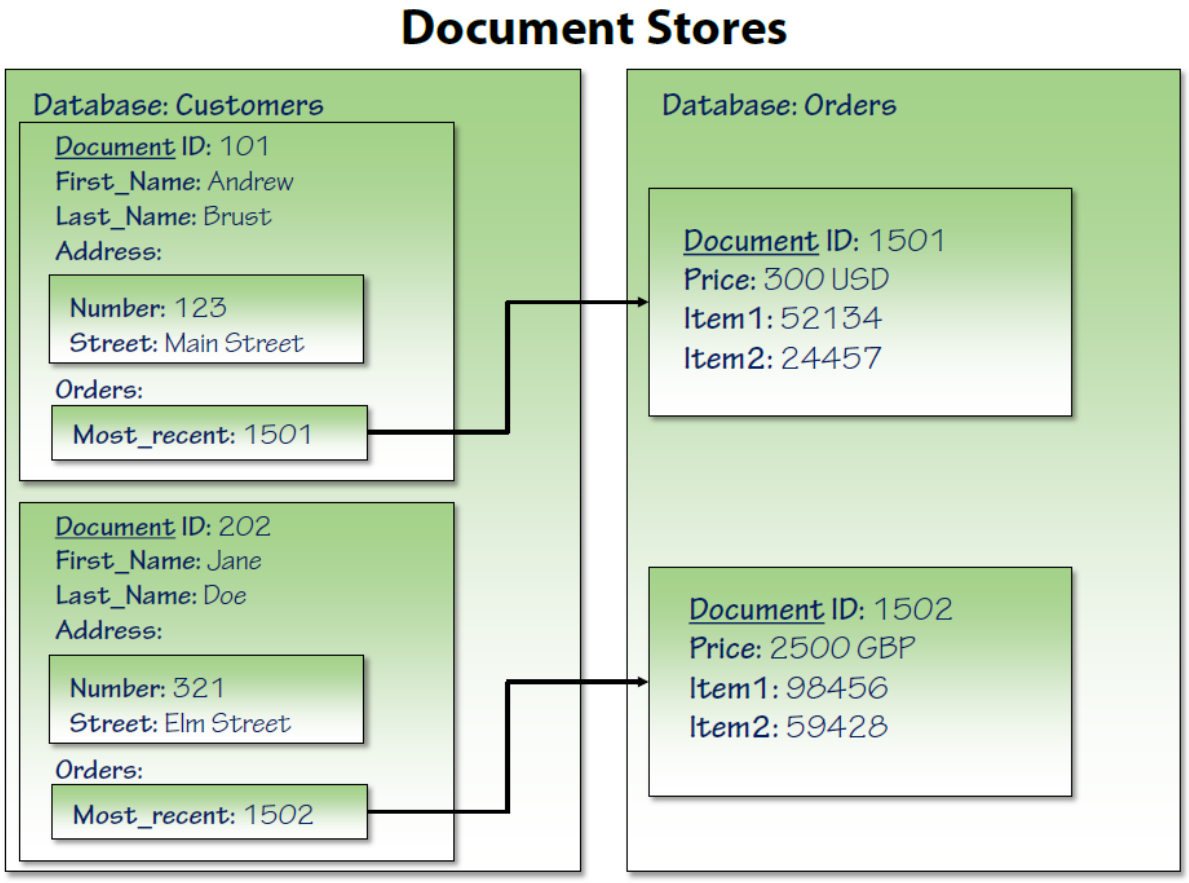
Relational model	Document model
Tables	Collections
Rows die telkens dezelfde velden in dezelfde volgorde bevatten	Documents waarbij het aantal velden en hun volgorde telkens kan verschillen
Columns	Key/value pairs

The following example shows a possible JSON representation describing a person.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

In je verdere opleiding zal er vooral aandacht besteed worden aan deze Document Databases waarin JSON-objecten worden opgeslagen. Hiervoor zal MongoDB gebruikt worden.

Zie Pluralsight: https://app.pluralsight.com/player?course=understanding-nosql&author=andrew-brust&name=understanding-nosql-m1-tech-breakdown&clip=1&mode=live&start=168.972099&noteid=f255f879-b8c9-48c6-808b-ee9fc52d6191



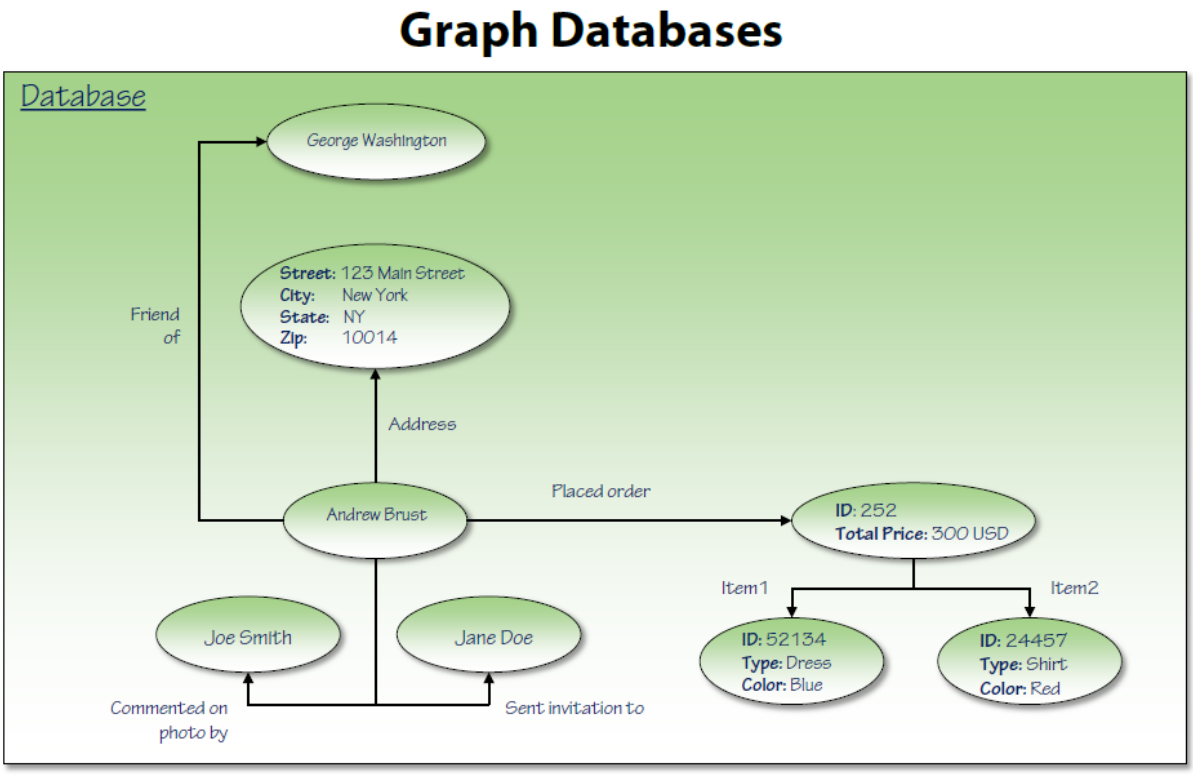
Demo CouchDB in PluralSight: https://app.pluralsight.com/player?course=understanding-nosql&author=andrew-brust&name=understanding-nosql-m1-tech-breakdown&clip=2&mode=live (12 min)

5.3.4. Graph stores

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.

Zie Pluralsight: https://app.pluralsight.com/player?course=understanding-nosql&author=andrew-brust&name=understanding-nosql-m1-tech-breakdown&clip=4&mode=live&start=77.868176&noteid=fb45d5c9-4e66-4d99-8b73-352b6c0de7e6



Een groot aantal bedrijven gebruikt NoSQL-databases.

Enkele namen: Google, Facebook, Adobe, LinkedIn, Mozilla,….

6. NoSQL, Relational or Both?

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.

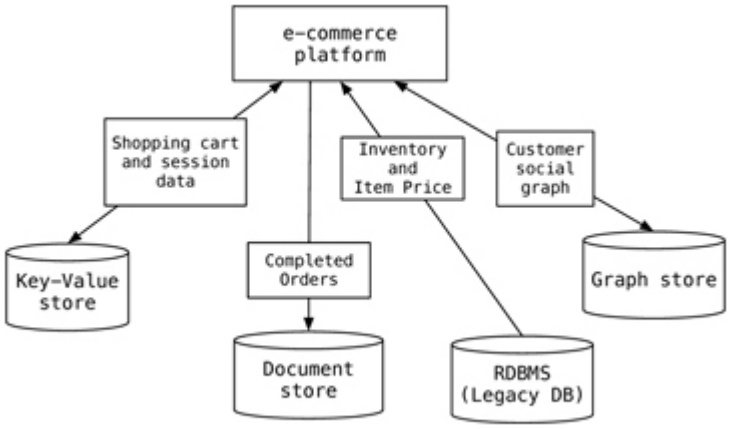


Figure 13.3. *Example implementation of polyglot persistence*

Zie Pluralsight: https://app.pluralsight.com/player?course=understanding-nosql&author=andrew-brust&name=understanding-nosql-m5-both&clip=3&mode=live&start=1.257044&noteid=35a1c93e-d59d-4be8-b130-a9928583f170

Recommendations

- **Large, public, content-centric properties: NoSQL**
- **Internal, LOB supporting business operations: relational**
- **Investment in RDBMS licenses, infrastructure, skills:**
  - Relational
  - Use both (application-dependent)
  - Use hybrid approaches
- **Productivity**
  - Do cost-benefit analysis
    - How much extra dev time/$$?
    - What is cost of less scalable system?
- **It will be tempting to use one for the other**
  - And it very well may work, but that doesn't make it right

Bibliografie.

http://www.sas.com/en_us/insights/analytics/big-data-analytics.html

Pluralsight cursus Big Data: the big picture - https://app.pluralsight.com/library/courses/bigdata-bigpicture/table-of-contents

Pluralsight cursus Understanding NoSQL - https://app.pluralsight.com/library/courses/understanding-nosql/table-of-contents

http://www.w3resource.com/mongodb/nosql.php

http://www.sas.com/en_us/insights/analytics/big-data-analytics.html#dmtechnical

"Succes met Big Data" – Wiebe van der Zee en Bert van der Zee – Uitg. VAN DUUREN MEDIA – April 2016 – 87 blz

http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf

https://en.wikipedia.org/wiki/Database_transaction

https://nl.wikipedia.org/wiki/Hashtabel

https://en.wikipedia.org/wiki/Hash_table

https://en.wikipedia.org/wiki/Associative_array

https://en.wikipedia.org/wiki/Column-oriented_DBMS

http://teachmehana.com/row-store-vs-column-store/