Introductie Databases

Ir. Ing. Bert Dingemans

bert@data-docent.nl

INHOUDSOPGAVE

Inleiding

In dit whitepaper worden databases beschreven. Databases zijn niet weg te denken binnen datagedreven werken. Wil je een beeld krijgen van welke soorten databases er zijn bekijk dan dit whitepaper.

Dit whitepaper is een onderdeel van meerdere whitepapers over data gedreven werken en data management. Databases zijn onder andere gericht op relationele databases en NoSQL databases. Daarnaast zijn er whitepapers over datamodelleren in deze serie van whitepapers, zie https://data-docent.nl.

DATABASES

Inleiding

In het vorige hoofdstuk hebben we gezien dat data beschouwd kan worden als een productiemiddel. Dat houdt dan vervolgens in dat we data moeten managen zoals we ook Assets - , Human Resources -en Financial management implementeren. Data Management implementeren kan het beste gedaan worden via een gestructureerde aanpak.

Nu is het mogelijk om dit voor de eigen organisatie uit te werken. Waarschijnlijk is het beter en efficiënter om gebruik te maken van een bestaand Data Management Framework. In dit hoofdstuk gaan we enerzijds in op het managen van data anderzijds gaan we kijken naar een centraal begrip wat hierbij hoort namelijk databases.

Data en databases zijn beiden een begrip en kunnen waarschijnlijk niet zonder elkaar bestaan. De data zit in de database en er zijn een aantal mechanismen die ervoor zorgen dat we de data kunnen inzetten in bedrijfsprocessen op momenten dat we die data nodig hebben.

In dit hoofdstuk zullen we een keten van IJssalons gebruiken, Alberto genaamd, als hulpmiddel om voorbeelden van de concepten te gebruikten

Definities

Bij databases kun je gebruik maken van een aantal definities om de kenmerken van bij databases gerelateerde begrippen te verklaren. Hieronder een aantal op wikipedia gebaseerde definities.

DATABASE

Een database, gegevensbank of databank is een (meestal digitaal opgeslagen) gegevensverzameling, ingericht zodat data opgeslagen, beheerd en geraadpleegd en gebruikt kunnen worden. Vermeldenswaardig is dat er een (groot) tijdsverschil kan zijn tussen het moment waarop de data opgeslagen en gebruikt wordt. Databases spelen een belangrijke rol bij het archiveren van feiten die in de werkelijkheid of context hebben plaatsgevonden. Daarnaast is het actueel houden van gegevens in een database een belangrijk aspect. De werkelijkheid kan veranderen en dat betekent dat de representatie van de data in een databank mee verandert dient te worden.

Databases zijn breed toegepast, vrijwel iedere organisatie, maar ook in de prive situatie worden databases ingezet ter ondersteuning van het functioneren van een organisatie maar ook het ondersteunen van de prive activiteiten.

In een IJssalon van Alberto worden ijsjes verkocht met behulp van een Kassa systeem (PoS). Alle verkooptransacties worden middels een kassa geregistreerd en deze transacties worden in een onderliggende database opgeslagen. Later worden deze transactie data geanalyseerd om te bepalen hoe de verkoopresultaten van de vestigingen zijn geweest.

Database Management Systeem

Uit het toepassen van databases ter ondersteuning van werkprocessen blijkt dat data in een database een productiemiddel is met waarde. Zo zal een organisatie dat dan ook willen behandelen. Met een databasemanagementsysteem (DBMS) wordt een systeem aangeduid dat als database opgeslagen gegevens ontsluit, bewaakt en beheert en ervoor zorgdraagt dat de waarde van data op het gewenste niveau blijft.

Een database bestaat veelal uit drie onderdelen:

- de opgeslagen gegevens (in één of meer bestanden),
- het programma waarmee de gegevens (centraal) worden onderhouden (server)
- eventueel een of meerdere gebruikersomgevingen (clients) die het gebruikers mogelijk maakt om de gegevens te behandelen.

Meestal is er één DBMS actief voor meer dan een gebruiker. Ook hier blijkt de waarde van de data, door deze centraal te beheren is de data die de werkelijkheid afbeeldt zodat alle gebruikers hetzelfde beeld zien in de data.

Bekende en veelgebruikte programma's zijn relationele DBMSen (afgekort tot rdbms) zoals Microsoft Access, SQLite, MySQL, Microsoft SQL Server en Oracle Database.

Bij de Alberto ijssalons zijn er kassasystemen waarin de verkooptransacties geregistreerd worden. Daarnaast is er een ERP systeem met daarin de voorraadadministratie aanwezig over de grondstoffen en de geproduceerde ijssoorten. Hierdoor wordt er zorggedragen dat iedereen hetzelfde beeld heeft van de werkelijkheid bij Alberto IJssalons. Dat is noodzakelijk voor een organisatie. Als een klant tien aardbeien ijsjes koopt (via een kassasysteem) dan heeft dat invloed op de voorraad aardbeienijs. Als een ijsmaker nieuw aardbeien ijs maakt dan heeft dat een positief effect. Daarnaast kan Alberto op basis van de verkoophistorie de productie- en verkoop van ijs voorspellen en op basis daarvan een planning maken voor de ijsproductie.

OLTP

Online transactieverwerking (OLTP) is een type databasesysteem dat wordt gebruikt in transactiegerichte toepassingen, zoals veel operationele- of administratieve systemen. Daarmee dus veelal ter ondersteuning van bedrijfsprocessen in een organisatie en daarmee feitelijk de gebeurtenissen in de werkelijkheid in de data vastleggen.

"Online" verwijst dat transacties die gedaan worden digitaal zijn en realtime verwerkt worden in de onderliggende database. Hierbij is het kenmerkend dat de data in kleine bewerkingen in de database gemuteerd worden. Dit stelt specifieke eisen aan het datamodel. Een datamodel beschrijft de structuur van de data waarmee de bedrijfsprocessen ondersteunt worden. Kenmerkend hierbij is dat gepoogd

wordt data slechts eenmalig op te slaan en dus duplicaten te voorkomen. In een volgend hoofdstuk gaan we nader in op deze datamodellen. De term staat in contrast met online analytische verwerking (OLAP), die zich in plaats daarvan richt op data-analyse (bijvoorbeeld planning- en managementsystemen). OLAP en OLTP zijn echter nauw aan elkaar gerelateerd, de een is de bron van de ander en omgekeerd.

Bij Alberto IJssalons is de onderliggende database gebaseerd op een OLTP model voor de administratieve systemen voor de verkoop- en voorraadadministratie. Administratieve systemen zoals ERP systemen zijn veelal gebaseerd op het OLTP model.

OLAP

Online analytical processing of OLAP is een data architectuur met bijbehorend datamodel dat data analyse ondersteunt. Het is een datamodel dat geïmplementeerd kan worden in een datawarehouse Business Intelligence tooling. Het kenmerk is dat de feiten uit de werkelijkheid kunnen worden geanalyseerd en langs dimensies in de organisatie kunnen worden ingedeeld en gegroepeerd.

De belangrijkste toepassingen van OLAP zijn het verkrijgen van inzichten door karakteristieken van de organisatie en organisatie onderdelen cijfermatig te analyseren. Cijfermatig betekent het doen van berekeningen zoals tellingen, totaliseren en het bepalen subtotalen op basis van dimensies in de data. Om de problemen van een moderne onderneming te kunnen oplossen, heb je speciale databaseschema's nodig.

Bij Alberto is een datawarehouse aanwezig waarin de data vanuit de administratieve toepassingen zoals de kassa- en voorraadsystemen worden ingelezen in de onderliggende database verrijkt met gegevens vanuit een aantal externe bronnen zoals klimaatdata en demografische data.

Het datawarehouse kent een bijzonder datamodel dat een dimensioneel datamodel wordt genoemd. Dit is opgebouwd uit data rond feiten en dimensies om die feiten te kunnen interpreteren. Hiervoor gebruikt Alberto een dashboard waarin hij met behulp van een aantal datavisualisaties zoals grafieken en taartdiagrammen de situatie van de organisatie via dimensies eenvoudig kan interpreteren. Daarmee krijgt hij inzichten uit de data en kan hij gewogen besluiten nemen.

RELATIONELE DATABASE

Een relationele database is een veelvoorkomende soort database. Het is opgebouwd volgens een relationeel model, dit model wordt nader uitgewerkt in een volgend hoofdstuk. Een relationele database wordt gekenmerkt door een duidelijke vooraf gedefinieerde structuur Het database management systeem zorgt dat de data in de database altijd aan de vooraf gedefinieerde structuur blijft voldoen.

De gegevens worden opgeslagen in tabellen waarin de rijen de soortgelijke groepen informatie, de records vormen, en de kolommen de informatie die voor elk record moet worden opgeslagen. Verschillende tabellen kunnen met elkaar worden verbonden door een kolom toe te voegen waarin een verwijzing naar een record in een andere tabel wordt opgenomen.

RELATIONELE DATABASES

Inleiding

Het relationele model voor databasemanagement is een gegevensmodel dat wordt gebruikt om gegevens in een relationele database te ontwerpen en te beheren. Dit model werd in 1970 geïntroduceerd door Edgar F. Codd en is sindsdien het meest gebruikte gegevensmodel voor moderne databases.

Het relationele model is gebaseerd op de relationele algebra zie voor een uitwerking https://nl.wikipedia.org/wiki/Relationele_algebra. Hierbij wordt gesteld dat een relatie bestaat uit een aantal kenmerken en vervolgens dat er tupels of records zijn en dat zijn combinaties van kenmerken. Hieruit kun je tabellen samenstellen. De kenmerken zijn de kolommen (met een beschrijving door de kolomkop) en de rijen zijn de tupels.

In het relationele model worden alle gegevens weergegeven in termen van relaties (ook wel tabellen genoemd) waarin de rijen de soortgelijke groepen informatie vormen (records) en de kolommen de informatie bevatten die voor elk record moet worden opgeslagen.

Het doel van het relationele model is om een declaratieve methode te bieden voor het specificeren van data en query's: gebruikers geven direct aan welke informatie de database bevat en welke informatie ze willen ophalen of toevoegen. Hierbij zie je dat er een datamodel ontstaat dat de predicaten beschrijft, het datamodel draagt er daarbij zorg voor dat alle gebruikers het relationele model op dezelfde wijze interpreteren.

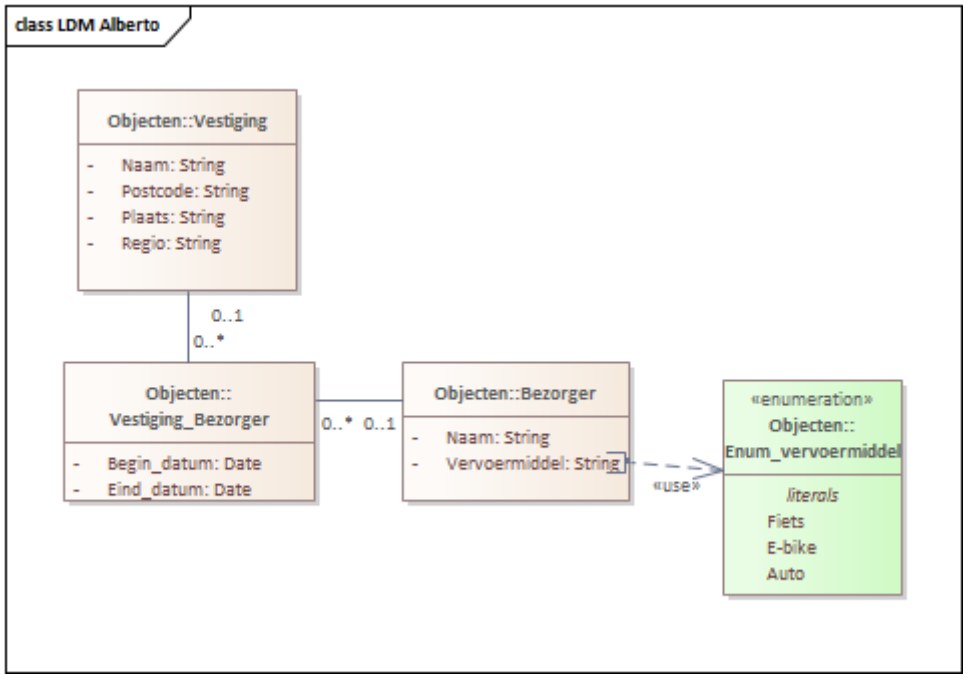
Het databasemanagementsysteem-software zorgt voor het inrichten van de datastructuren in de vorm van tabellen en de ophaalprocedures voor het beantwoorden van de query's. Kortom, het relationele model in een relationele database biedt een beschreven structuur, nauwkeurigheid, gebruiksgemak, samenwerking, beveiliging en categorisatie voor het beheren van databases. Hiervoor is in de loop der jaren een standaard ontstaan en dat is een taal die het mogelijk maakt om het relationeel model te beschrijven en te gebruiken. Dat is de Structured Query Language (SQL), zie hiervoor de volgende paragraaf.

In de Alberto IJssalons wordt met data in het relationeel model gewerkt. Zo is er een tabel met data over de bezorger. De bezorger tabel heeft een aantal kenmerken zoals voornaam, tussenvoegsel, achternaam en vervoermiddel. In een tabel zie je vervolgens dat er meerdere records zijn die de kenmerken invullen met waarden eventueel gebaseerd op een domein. Zo kun je bij een vervoermiddel alleen kiezen voor auto, fiets en ebike.

Voornaam	Tussenvoegsel	Achternaam	Vervoermiddel
Petra		Hubert	E-bike
Jan	De	Vries	Auto

Voor alle relaties kunnen we een tabel definiëren en deze tabellen hebben onderling ook relaties conform de relationele algebra. Bijvoorbeeld een Vestiging waar een Bezorger voor werkt. Zo ontstaat een model

in het relationele model in een grafische representatie. In de volgende hoofdstukken gaan we deze modellen uitwerken, maar hieronder een voorbeeld.



Structured Query Language

SQL (Structured Query Language) is een ANSI/ISO-standaardtaal voor een relationeel databasemanagementsysteem (RDBMS). Het is een gestandaardiseerde taal die gebruikt kan worden voor taken zoals het bevragen en het aanpassen van gegevens in een relationele database die gebaseerd is op de relationele algebra. SQL kan met vrijwel alle moderne relationele databaseproducten worden gebruikt.

De taal is gebaseerd op commando's. Commando's zijn in SQL op tekst gebaseerd en hebben een formeel karakter. Formeel houdt in dat er een syntax is voor de taal en dat commando's strikt aan de formele specificaties dienen te voldoen. Is dat niet het geval dan zal de database een foutmelding genereren. Hiermee blijkt dat SQL nauw verwant is aan programmeertalen als Python of C#.

De structured query language kan opgedeeld worden in een aantal domeinen die als subtalen gedefinieerd zijn. Van iedere subtaal geven we een korte beschrijving en daarbij een aantal voorbeelden van een commando over de

Data Definitie Taal (DDL), het definiëren van de structuren van de relaties die geïmplementeerd worden in tabellen in en views. Views zijn een hulpmiddel om weergaven te maken op basis van de structuur in de tabellen.

Voorbeelden van de DML zijn:

Creëren van een tabelstructuur

```
CREATE TABLE [54707_sa].[Bezorger](
	[Bezorger_id] [int] IDENTITY(1,1) NOT NULL,
	[Naam] [varchar](50) NOT NULL,
```

```
      [Vervoermiddel] [varchar](50) NULL,
 CONSTRAINT [PK_Bezorger] PRIMARY KEY CLUSTERED
([Bezorger_id] ASC)
```

Aanmaken van een weergave voor data uit een of meerdere tabellen

```
CREATE OR ALTER VIEW [BezorgerExtra] as
SELECT Bezorger.Naam AS Bezorgernaam
, Bezorger.Vervoermiddel
, Vestiging.Naam AS Vestigingnaam
, Vestiging.Postcode
, Vestiging.Plaats
, Vestiging.Regio
, Vestiging_Bezorger.Begin_datum
, Vestiging_Bezorger.Eind_datum
, Vestiging_Bezorger.Vestiging_bezorger_id
FROM Bezorger
INNER JOIN Vestiging_Bezorger ON Bezorger.Bezorger_id = Vestiging_Bezorger.Bezorger_id
INNER JOIN Vestiging ON Vestiging_Bezorger.Vestiging_id = Vestiging.Vestiging_id
```

Data Manipulatie Taal (DML), is het manipuleren van de data, dus de inhoud van de tabellen. Hierbij spelen de rijen en kolommen in een tabel een centrale rol. Je dient de kolommen en rijen die gemanipuleerd gaan worden definiëren in de commando's voor manipulatie. Er zijn drie soorten manipulaties namelijk

Toevoegen van een rij aan de bezorger tabel

```
INSERT INTO [Bezorger]
           ([Naam]
           ,[Vervoermiddel])
     VALUES
           (<Naam, varchar(50),>
           ,<Vervoermiddel, varchar(50),>)
```

Update de inhoud van één rij in de bezorger tabel

```
UPDATE [Bezorger]
   SET [Naam] = <Naam, varchar(50),>
      ,[Vervoermiddel] = <Vervoermiddel, varchar(50),>
 WHERE Bezorger_id = <Bezorger_id>
```

Verwijderen van één rij uit de bezorger tabel

```
DELETE FROM [Bezorger]
WHERE Bezorger_id = <Bezorger_id>
```

Data Control Language (DCL) heeft betrekking op het bepalen van de toegang tot de data in de tabellen, vanuit privacy maar ook van bedrijfskundig perspectief heeft niet iedereen toegang tot dezelfde data in rijen en kolommen. Dat kan gedaan worden op de structuur van de data, schema's, tabellen en views.

```
REVOKE SELECT ON OBJECT::BeveiligingDemo.BeveiligingTabel1 TO BeveiligingGebruiker;

GRANT SELECT ON OBJECT::BeveiligingDemo.BeveiligingTabel2 TO BeveiligingGebruiker;
```

```
REVOKE INSERT, UPDATE, DELETE ON OBJECT::BeveiligingDemo.BeveiligingTabel2 TO
BeveiligingGebruiker;
```

Data Query Language (DQL), data die in databases opgeslagen worden wil je er op enig moment weer uithalen, dat wordt gedaan met de query language waar de naam SQLop gebaseerd is. Hierbij dient er gekeken te worden voor welke doelen data uit de data opgehaald wordt met queries. Dat kan ter ondersteuning van transacties, of wel administatieve handelingen zijn (OLTP), maar ook voor het krijgen van inzicht en op basis daarvan kenniswerkers te ondersteunen (OLAP).

OLTP gebaseerde query

```
SELECT [Bestelling_id]
      ,[Bestel_datum]
      ,[Bestel_aantal]
      ,[Bestel_bedrag]
      ,[Postcode]
      ,[Product_categorie]
      ,[vestiging_id]
FROM [Bestelling]
WHERE [Bestelling_id] = 100
```

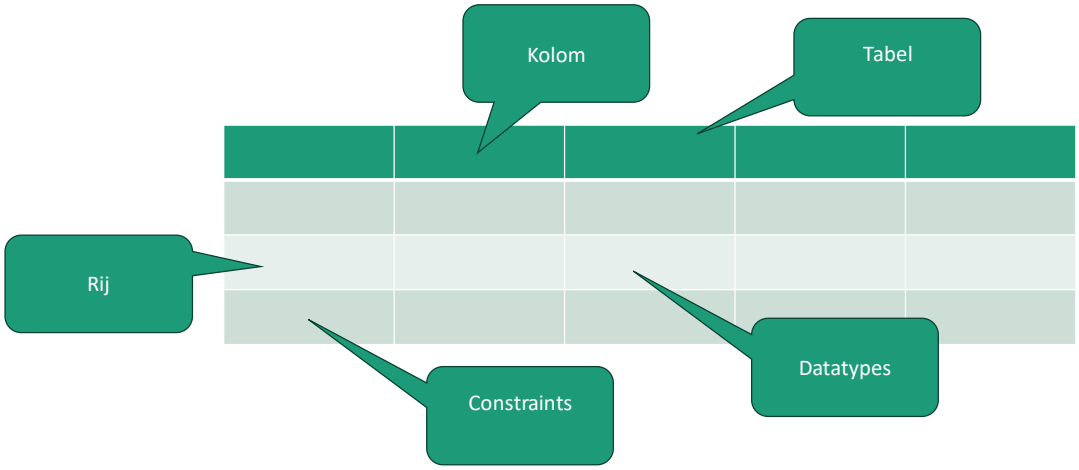
OLAP gebaseerde query

```
SELECT Sum(Bestel_bedrag) as Omzet
      ,[Ggemeente]
      ,[DMaand]
        FROM [DataMart_Bestelling_OLAP]
GROUP BY [Ggemeente]
,[DMaand]
```

Tabellen, kolommen en datatypes

Tabellen zijn het basisconcept in relationele databases. De relaties worden op basis van kenmerken uitgewerkt en vervolgens worden er records aan een tabel toegevoegd die ieder een eigen vulling hebben op basis van de kolommen dat bestaat uit combinaties van waarden.

Onderstaande afbeelding geeft een eenvoudige weergave van de kenmerken van een tabel



Kolommen, vormen de basis en beschrijven de eigenschappen van de entiteit die in een tabel wordt opgeslagen. Kolommen zijn daarmee een vorm van metadata die daarmee een beschrijving geven van wat het kenmerk inhoudt. Daarnaast legt de kolom een aantal beperkingen op aan de data die in de rijen bij deze kolom worden opgeslagen. Zoals reeds gezegd is een relationele database gebaseerd op structurele data en dwingt een database de regels af bij het bewerken van de gegevens.

In een relationele database is dus de structuur bepaald door de kolom echter ook de beschrijving van zowel de tabel als de kolommen is een belangrijk kenmerk van een kolom. In databases zijn daarom mogelijkheden opgenomen om een beschrijving of definitie te geven van zowel de tabel als de kolom. In de datamodellen vind je deze beschrijvingen vaak op een toegankelijke manier terug. Hiermee kun je ervoor zorgdragen dat iedere gebruiker van de data uit de tabellen kan beschikken over dezelfde definities.

Datatypen, de kolommen hebben allemaal een datatype. Met deze datatype bepaal je de structuur en perk je deze in naar het gewenste datamodel voor de eigen organisatie. De datatypen hebben een eigen structuur en kenmerken en bepalen hiermee op welke wijze de data wordt vastgelegd en met welke beperkingen. Bij het kiezen van de datatypen dient altijd de performance van de database meegenomen te worden. Met name in databases van grote omvang telt de omvang van de opgeslagen data in een kolom een belangrijke rol bij de performance.

Bij het ontwerpen van een databases met behulp van een datamodel dient dus bepaald te worden wat het juiste datatype is in de eigen context van de toepassing van de data. Iedere databases heeft een eigen opzet gemaakt van de datatypen, echter in de basis is de volgende indeling toegepast:

- **Alfanumeriek**, combinatie van letters, getallen en soms ook bijzondere letters gebaseerd op de talen in de wereld (Chinees, Noors, etc)

    - **Vast aantal karakters**, vooraf bepaald aantal alfanumerieke karakters, soms verder ingeperkt met een invoermasker. Bijvoorbeeld een Nederlandse postcode heeft een vast formaat van vier cijfers een spatie en twee letters, in totaal dus 7
    - **Variabel aantal karakters**, hierbij wordt een maximaal aantal beschikbare karakters bepaald maar als de ruimte niet nodig is bij opslag dan wordt die niet gebruikt. Denk bijvoorbeeld aan voornamen van Rik en Marie-Antoinette. Voor de eerste worden slechts drie karakters gebruikt voor de ander vijftien
    - **Karakters op basis van lokalisatie,** De Engelse taal kent een beperkt aantal karakters in het alfabet andere talen kennen meer karakters en dat is per taal anders denk aan de ö de é of è.
    - **Vrije tekst,** vooraf niet te bepalen hoeveelheid tekst van drie letters tot honderden pagina's tekst.
- **Numeriek**, getallen voor het doen van berekeningen maar ook het voeren van (financiële)registraties
    - **Gehele getallen**, gehele getallen onder en boven nul binnen een bepaald bereik. Bijvoorbeeld het aantal elementen dat besteld wordt
    - Decimale getallen, binnen een bepaald bereik dat vooraf gedefinieerd wordt. Hierbij kunnen ook getallen achter de komma geregistreerd worden. Bijvoorbeeld de registratie van valuta bedragen.
    - **Real of float**, getallen binnen een heel groot bereik van zeer grote en kleine getallen. Hierbij kan er een afronding van de getallen plaatsvinden. Bijvoorbeeld registratie van de getallen 150.000.000.000.000 maar ook 0,00000000000000045
- **Logisch,** indeling in waar of niet waar, ja en nee
    - **Boolean,** True of False bijvoorbeeld voor registratie of een klant korting krijgt of niet
- **Datum**, datum en tijd registratie op basis van kalenderdatum
    - **Kalenderdatum**, registratie van de kalanderdatum op basis van jaar,maand en dag. Bijvoorbeeld de datum waarop een product besteld is
    - **Kalenderdatum en -tijd**, kalenderdatum uitgebreid met tijd in uren, minuten seconden en milliseconden. Bijvoorbeeld de bezorgdatum en tijd bij een thuisbezorgd registratie
    - **Short datum**, vaak zetten datums binnen een bepaalde range bijvoorbeeld tussen 1900 en 2060, voor veel datumvelden is dat voldoende. Bijvoorbeeld de besteldatum
- **Overigen**, veelal alfanumerieke data met extra beperkingen gebaseerd op een taal of standaard
    - **Blob,** Binary Large Object Block, opslag van binaire data
    - **XML**, Extensible Markup Language, gebaseerd op een XML taal voor het combineren van data met beschrijvende metadata
    - **JSON**, Javascript Object Notation vergelijkbaar met XML maar gebaseerd op een andere standaard.

Check Constraints, naast de datatypen is het mogelijk om bepaalde extra checks te doen binnen een bepaald datatypen. Bijvoorbeeld bepaalde waarden die moeten gelden op basis van een conditie. Een check op basis van leeftijd of iemand een alcoholhoudend ijsje kan bestellen.

Structuur bewaken

Structuur in de data is binnen een relationele database. Dat is geïmplementeerd in de kolommen en de datatypen. In de vorige pragraaf zijn we ingegaan op kolommen en de beperkingen die daarop kunnen gelden op basis van een datatype of check constraints. Werk je deze regels rond de datatypen vergaand uit dan heeft dat een aantal belangrijke voordelen. Met name dat de database op basis van de ingestelde regels de inhoud van de tabellen gaat valideren op basis van deze regels is kenmerkend. Hiermee kun je feitelijk een deel van de registratie en de bijbehorende validaties naar het database management systeem verplaatsen. Dat is met name voor gebruikers, maar ook voor programmeurs van data gedreven software een groot voordeel.

SLEUTELS

Naast de reeds genoemde validaties zijn er twee kenmerkende extra validaties en controlefuncties aanwezig in relationele databases. Binnen databases wordt data opgeslagen in tabellen. Echter veelal wordt de data opgeslagen in meerdere tabellen en zijn er ook relaties van belang over de tabellen heen. Denk bijvoorbeeld aan klanten en bestellingen die een relatie hebben. De bestelling wordt gedaan door een klant.

Om ook hier een structuur te implementeren en af te dwingen zijn er een aantal extra concepten opgenomen namelijk de primaire- en de verwijzende sleutel. Deze lichten we kort toe in de sub paragrafen.

PRIMAIRE SLEUTEL

Een primaire sleutel in een database is een unieke identificatie voor elke rij in een tabel. Het zorgt ervoor dat elke rij zijn eigen unieke set van waarden heeft, waardoor je gemakkelijk gegevens kunt vinden en manipuleren. Je kunt zelf een combinatie van kolommen in een tabel kiezen waarmee iedere rij uniek wordt bepaald. Maar meestal is dit een sleutel veld met extra kenmerken. Bijvoorbeeld dat bij toevoegen van een nieuwe rij er een automatisch opgehoogd nummer wordt aangemaakt en in het sleutelveld wordt geplaatst.

Het is belangrijk dat geen twee rijen dezelfde primaire sleutel hebben, omdat dit de integriteit van de database zou schaden. Ook hierbij een voorbeeld dat je een registratieve taak rond de data manipulatie in het database managementsysteem belegt.

Je kunt een primaire sleutel definiëren wanneer je een tabel maakt met SQL, zoals in het volgende voorbeeld:

```
ALTER TABLE Felicitatie
ADD Felicitatie_id INT NOT NULL Identity(1,1)

ALTER TABLE [Felicitatie]
 ADD CONSTRAINT [PK_Felicitatie]
PRIMARY KEY CLUSTERED ([Felicitatie_id] ASC)
```

Het toevoegen van een primaire sleutel die geen betekenis heeft wordt veel toegepast bij het ontwerpen van relationele databases. Denk bijvoorbeeld aan een personeelsnummer, een artikelnummer of ordernummer. Allemaal voorbeeld van de primaire sleutel en vervolgens als verwijzende sleutel wordt gebruikt.

VERWIJZENDE SLEUTEL

Een verwijzende sleutel is een set van één of meer kolommen in een database tabel die verwijst naar de primaire sleutel van een andere tabel. Het doel van een verwijzende sleutel is om de integriteit van de gegevens te waarborgen door ervoor te zorgen dat de waarden die het bevat, overeenkomen met de waarden in de primaire sleutel van de gerelateerde tabel. Dit betekent dat de verwijzende sleutel een relatie creëert tussen twee tabellen, wat essentieel is voor het relationele database model.

Ook hierbij wordt de registratieve verantwoordelijkheid hiermee in het database management systeem gebracht. Het DBMS gaat controleren bij mutaties in de primaire of verwijzende sleutels of dit wel conform de regels is bij de sleutels die gebruikt worden om de relatie tussen de tabellen consistent te houden

Bijvoorbeeld, als je een tabel hebt met vestiging waar de bestelling is gedaan s en een andere tabel met bestellingen, dan kan de bestellingen tabel een verwijzende sleutel bevatten die verwijst naar de primaire sleutel van de vestiging. Dit zorgt ervoor dat elke bestelling gekoppeld is aan een specifieke vestiging en dat de database dit afdwingt.

Vestiging

vestiging_id	naam	plaats
1	Oude gracht	Utrecht
2	Overvecht	Utrecht
3	Vreeswijk	Nieuwegein
4	Vianen	Vijfheerenlanden
5	Leerdam	Vijfheerenlanden
6	Geldermalsen	West Betuwe
7	Culemborg	Culemborg

Bestelling

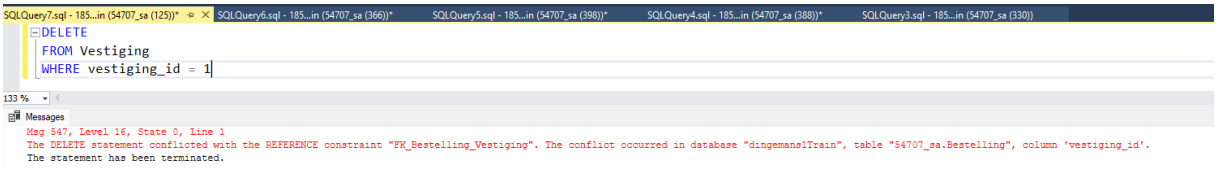
Bestelling_id	Bestel_datum	Bestel_aantal	Postcode	Product_categorie	vestiging_id
1	2022-04-03	6	4102AA	Softijs	1
2	2022-04-04	1	4112HB	Koffie	1
3	2022-04-05	1	4116RN	IJs	1
4	2022-04-06	1	4196RS	Softijs	1
5	2022-04-07	2	4142ZZ	Koffie	1
18	2022-04-04	1	4196RS	IJs	1
19	2022-04-05	1	4142ZZ	Softijs	1

In het bovenstaande voorbeeld zie je hoe de tabellen met behulp van de primaire sleutel in de vestiging tabel als verwijzende sleutel in de bestelling tabel.

Hieronder vervolgens hoe je op basis van SQL je een verwijzende sleutel kunt definiëren:

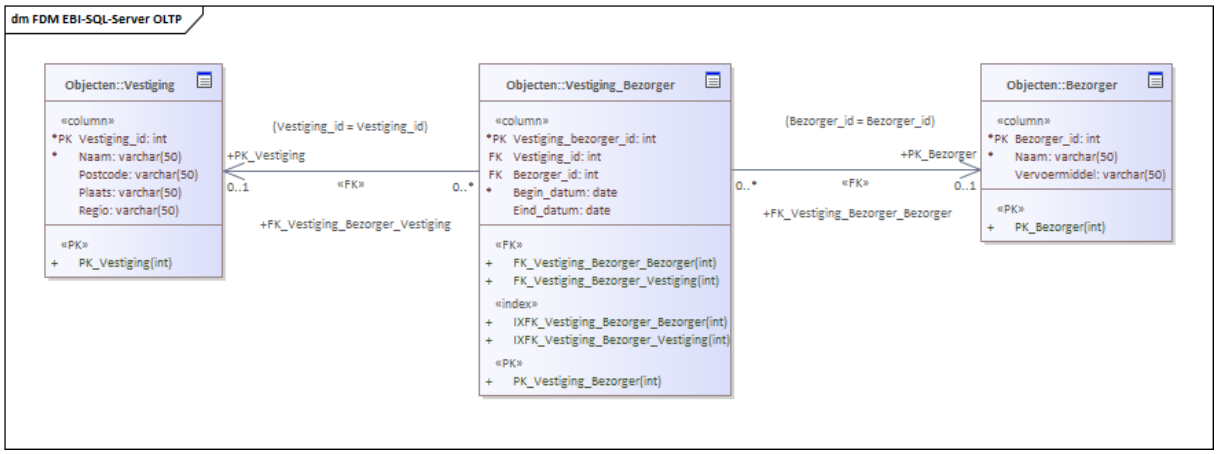
```
ALTER TABLE [Bestelling]
ADD  CONSTRAINT [FK_Bestelling_Vestiging]
FOREIGN KEY([vestiging_id]) REFERENCES [Vestiging] ([Vestiging_id])
```

Zijn de foreign en primary keys op de juiste manier ingesteld dan gaat de database dit bewaken. Hieronder een eenvoudig voorbeeld in een willekeurige interactie hoe de database hier de controles uitvoert en de gebruiker attendeert dat er een inconsistentie zou ontstaan en daarom het commando niet uitvoert.



FYSIEK DATAMODEL EN SLEUTELS

In de voorgaande paragrafen zijn een aantal voorbeelden gegeven van hoe de database op basis van een structuur de inhoud van de tabellen op basis van consistentie regels bewaakt. Je kunt je voorstellen dat zeker bij omvangrijke databases er zeer veel regels zorgdragen voor het bewaken van de consistentie. Hier overzicht over houden is daarmee een complexe taak voor personen die een database beheren of gebruiken.



Daarom is er een visuele modelleerwijze ontstaan die inzicht geeft in de structuur binnen de database maar ook de regels die zorgdragen voor consistentie van de data. Hierboven een eenvoudig voorbeeld van een visueel model het ER diagram. ER diagram is de notatiewijze die hiervoor gebruikt wordt. In een later hoofdstuk is deze notatiewijze uitgewerkt.

ACID

In een DBMS zijn nog een aantal controlerende en ondersteunende functionaliteiten opgenomen die zorgdragen voor consistentie van de data in de database. Een mooie functionaliteit is ACID.

Stel dat je in een database een aantal bewerkingen doet in de database waarbij je de inhoud van tabellen gaat muteren. Deze serie van bewerkingen worden als een enkele bewerking beschouwd terwijl het in de database bestaat uit meerdere bewerkingen bestaat. In dat geval wordt ACID belangrijk. Hieronder een opsomming waar de afkorting voorstaat. Vervolgens een voorbeeld uitwerking

- **Atomic** (bewerkingen binnen de transactie slagen of mislukken altijd in zijn geheel, als dit mislukt wordt de database teruggebracht in de initiële toestand).
- **Consistant** (database heeft altijd consistente en (schema) valide data.

- **Isolation** (resultaat van een bewerking alleen zichtbaar na beëindiging van de transactie).
- **Durable** (na beëindiging van de transactie blijven de mutaties bewaard).

Stel een klant bestelt een paar ijsjes en wil dat die thuisbezorgd worden. Dit bestaat uit een aantal deelbewerkingen op de data in de tabellen

- Aanmaken van een bestelling
- Aanmaken van een bezorg registratie met het bepalen van de bezorger
- Doen van een betaling

Pas als al deze bewerkingen in de data correct zijn uitgevoerd worden deze bewerkingen definitief gemaakt en wordt de data vastgesteld. Want als één van de drie bewerkingen mislukt, zoals de betaling dan wil je dat alle eerdere bewerkingen ook teruggedraaid worden. Je wilt tenslotte niet dat er ijsjes gemaakt worden en vervolgens bezorgd worden als er niet voor betaald is. Dat is het kenmerk van ACID. Relationele databases met name voor OLTP (transactie verwerking) is in hoge mate afhankelijk van dit mechanisme.

CRUD

In de data control language wordt gedefinieerd welke rechten een gebruiker heeft met betrekking tot de data. Dit kunnen eindgebruikers zijn maar ook beheerders en desgewenst andere informatiesystemen kunnen toegang krijgen tot de data.

Daarbij dient er vanuit verschillende perspectieven beheer worden toegepast welke gebruikers wanneer en op welke wijze toegang heeft tot de data. Hierbij wordt gebruik gemaakt van CRUD.

CRUD staat voor de bewerkingen op de data

- **Create**, creëren van data, in SQL op basis van INSERT statements.
- **Read**, lezen van data uit een of meerdere tabellen. In SQL op basis van het SELECT statement.
- Update, muteren van de data in rijen van een of meerdere tabellen in SQL met het UPDATE statement.
- **Delete**, verwijderen van de data in rijen van een of meerdere tabellen met het DELETE SQL statement.
- **Overigen**, er zijn een aantal extra bewerkingen voor overige concepten in een relationele databases, denk hierbij bijvoorbeeld het executeren van logica in procedures en functies maar ook het toekennen van rechten zelf is een bewerking in een database.

CRUD kan worden toegepast op tabellen waarin de data is opgeslagen. Daarnaast zijn er schema's, dat is feitelijk een extra groepering van database entiteiten zoals tabellen en views en ook views kunnen ingezet worden voor het definiëren van de rechten die een gebruiker of gebruikersgroep op de entiteiten heeft.

CRUD heeft een modelleerwijze de CRUD matrix genaamd en ook dit is een eenvoudige visuele representatie die toegepast kan worden. In een volgend hoofdstuk beschrijven wij de CRUD matrix in detail. Hieronder een eenvoudig voorbeeld.

	Beheerder	Manager	Medewerker	Finance
Activiteit	CRUD	CRUD	CRU	
Vestiging	CRUD	R	R	R
Medewerker (data)	D	R		CRU
Salarisschaal	CRUD	R	R	CRU
Rooster	CRUD	CRU		

SCHEMA ON READ/WRITE

In dit hoofdstuk hebben we stilgestaan bij de aspecten rond de structuur van de data in een relationele database. Hierbij zien we dat het schema in een database door een RDBMS wordt gehandhaafd bij het schrijven van de data naar een database. De activiteiten behorend bij het handhaven worden daarmee belegd bij het RDBMS. Dat is daarmee het verplaatsen van dergelijke activiteiten die anders bij een gebruiker zouden liggen.

Een relationele database voldoet daarmee aan het paradigma **schema on write.** Schema on write impliceert daarmee een aantal kenmerken indien schema on write is toegepast:

- **Validatie bij schrijven naar de database**, de validatie van de data aan het schema vindt plaats tijdens het schrijven naar de database. Op basis van CRUD dus CUD acties.
- **Opslaan van inconsistente data is niet mogelijk,** door validatie van de data aan de structuur bij het schrijven betekent dit dat data die niet voldoet niet kan worden weggeschreven naar de database. Zonder correctie van die data gaat de inconsistente data daarmee verloren.
- **Performance van de database bij schrijven,** gezien de validatie van de inkomende data en het controleren van de consistentie duurt het wegschrijven van de data naar de database logischerwijs langer.

Aan de ene kant is het schema on write een krachtig paradigma voor de relationele database waar dit wordt toegepast, zeker in het kader van OLTP dataverwerking. Echter in een aantal gevallen kan dit paradigma ook als beperkend worden beschouwd. Met name dat data kan verdwijnen als het niet aan het schema voldoet kan in een andere context als een beperking gezien worden.

Schema on read is daarom het paradigma naast schema on write dat hiervoor geïntroduceerd wordt. Het houdt in dat een database de data die ontvangen wordt zonder schema validatie wordt opgeslagen in de database en dat als men die structuur in een later stadium bij het gebruiken van de data in een toepassing toetst in de data. Dus validatie van data vindt pas plaats op het moment dat de data gelezen wordt voor gebruik in een toepassing. Gevolg daarvan is dat er per gebruikstoepassing bepaald kan worden of het schema toegepast kan worden.

Schema on read heeft daarmee een aantal andere implicaties:

- Structuur in de data wordt op een later moment bepaald dus nadat het opgeslagen is.
- Data kan mogelijk ingezet worden voor meerdere gebruikstoepassingen.
- Evaluatie van de bruikbaarheid van de data vindt later plaats.
- Interpretatie van de data op basis van een datamodel is flexibel en vraagt overeenstemming.

Schema on read is een krachtig paradigma dat in bepaalde toepassingen voordelen biedt ten opzichte van schema on write. Echter vergeet niet dat schema on write bij OLTP toepassingen nog steeds de voorkeur verdiend.

In het volgende hoofdstuk gaan we in op NoSQL databases waarin het implementeren van Schema on Read mogelijk is.

BIG DATA

Is een organisatie data gedreven en worden er verschillende soorten data verzameld dan komt er een moment waarop het introduceren van Big Data relevant wordt. Dat ontstaat veelal op een natuurlijke wijze. Je kunt die evolutionaire ontwikkeling op z'n beloop laten, echter vanuit data management perspectief is richting geven aan deze ontwikkeling. Met data governance of -architectuur wil je sturing geven aan deze ontwikkeling. In dit hoofdstuk kijken we naar de kenmerken en dimensie van Big Data.

Definitie van big data

Op wikipedia is een mooie definitie te vinden:

> Men spreekt van big data wanneer men werkt met een of meer datasets die te groot zijn om met reguliere databasemanagementsystemen onderhouden te worden.
>
> Big data spelen een steeds grotere rol. De hoeveelheid data die opgeslagen wordt, groeit exponentieel. Dit komt doordat consumenten zelf steeds meer data opslaan in de vorm van bestanden, foto's en films (bijvoorbeeld op Facebook of YouTube, waar Facebook de door de gebruikers gewiste data toch bewaart) en organisaties, overheden en bedrijven steeds meer data over burgers produceren en opslaan, maar ook doordat steeds meer apparaten zelf data verzamelen, opslaan en uitwisselen (het zogenaamde internet der dingen). Hierdoor is er steeds meer sensordata beschikbaar.
>
> Niet alleen de opslag van deze hoeveelheden is een uitdaging. Ook het analyseren van deze data speelt een steeds grotere rol. Deze data bevatten immers een schat aan informatie voor verschillende doeleinden, zoals marketing, wetenschappelijk onderzoek, of preventief onderhoud.

Kenmerken Big Data

Big data heeft een aantal kenmerken die afwijken ten opzichte van de data in traditionele databanken. Hierbij een aantal gezichtspunten op de kenmerken.

REDENEN INZET BIG DATA

Hieronder een aantal redenen om big data te introduceren

- **Proces optimalisatie**, gebruik je nieuwe bronnen zoals logs van informatiesystemen en ga je die analyseren dan kun je mogelijk inzichten krijgen over de werkprocessen die geoptimaliseerd kunnen worden. Denk bijvoorbeeld aan het werkveld process mining.
- **Ondersteunen beslissingen**, wil je het nemen van beslissingen doen op basis van een combinatie van databronnen en ook big data bronnen gebruiken en deze data transformeren tot een datamodel dat gebaseerd is op meerdere parameters. Dan zal dat het nemen van beslissingen verbeteren. Denk bijvoorbeeld aan het werkveld social listening.
- **Nieuwe markten ontdekken**, gebruik je big data bronnen dan ontstaan er nieuwe mogelijkheden om producten en diensten te ontwikkelingen. De social media platformen zijn hier mooie voorbeelden van. Ondertussen niet meer weg te denken als bron van informatie en allemaal gebaseerd op big data

- **Mogelijkheid voor voorspellingen**, voorspellende algoritmen stellen hoge eisen aan de kwaliteit van de data. Door meerdere bronnen te combineren inclusief big data dan zal het doen van voorspellen verbeteren. Denk bijvoorbeeld aan de verkeersdiensten in je routeplanner.
- **Fout- of fraude detectie**, dit is vaak gebaseerd op afwijkingen in transacties die gedaan worden. Zeker als het aantal transacties groot is dan zal big data mogelijkheden bieden. Denk bijvoorbeeld aan spam filters in de email.
- **Wetenschappelijke ontdekkingen doen**, wetenschappelijk onderzoek doen is gebaseerd op datasets. Heb je grotere datasets dan kun je inzichten krijgen uit een grotere dataset die anders niet mogelijk zijn. Bijvoorbeeld in de genetica worden big data-technologieën gebruikt om enorme hoeveelheden DNA-sequentiegegevens te analyseren. Dit stelt onderzoekers in staat om genetische variaties en mutaties te identificeren die verband houden met bepaalde ziekten.

BIG DATA ENABLERS

Er zijn technologisch en maatschappelijk een aantal ontwikkelingen gaande die de ontwikkeling van big data stimuleren

- **Digitalisering**, steeds meer data wordt elektronisch ingewonnen en toegepast. Daarmee ontstaan er steeds meer big data sets. Denk bijvoorbeeld aan het digitaliseren van eeuwenoude archieven.
- **Nieuwe analyse en data science technieken**, het analyseren van big data sets is een uitdaging op het gebied van performance. Echter er ontstaan steeds meer data science toepassingen die gebaseerd zijn op gedistribueerde verwerking.
- **Betaalbaarheid soft- en hardware**, veel big data is beschikbaar als open source versie, daarnaast is het vanwege gedistribueerde verwerking te installeren op relatief eenvoudige hardware. Dat is een belangrijke kostenreductie. Denk hierbij bijvoorbeeld aan Hadoop.
- **Algoritmen voor gedistribueerde verwerking**, zijn datasets erg groot dan wordt het analyseren van deze dataset door één verwerkingseenheid een probleem. Big Data toepassingen zijn vaak gebaseerd op een cluster van verwerkingseenheden die met elkaar samenwerken, dan zal de performance enorm verbeteren. Denk bijvoorbeeld aan Map Reduce en de opvolgers van dit algoritme.
- **Sociale media**, is niet meer weg te denken uit de samenleving. Social media verzameld enorme hoeveelheden data slaat deze op en analyseert deze data voor het opstellen van profielen.
- **Hyper connected devices (IoT),** Internet of Things is gebaseerd op sensoren en deze produceren data over fenomenen in de omgeving of het ding. Ondertussen zijn er weinig dingen meer waar geen sensoren zitten en deze data wordt opgeslagen voor latere analyse. Denk aan dingen als vliegtuigen, (vracht)auto's, treinen of schepen.
- **Cloud computing,** steeds meer organisaties omarmen cloud computing en maken gebruik van de internationale cloud dienstverleners. Zij zijn in staat om big data toepassingen in de cloud te ondersteunen en aan te bieden. Denk aan AWS en Azure als grote cloud aanbieders.

VIJF VS

Doug Laney heeft in 2001 een artikel geschreven over de 3V's waarin hij de verschillen van big data ten opzichte van de traditionele dataverwerking verklaard. Later heeft Arcitura er twee Vs aan toegevoegd. Dat brengt ons tot de vijf Vs:

- **Volume** (hoeveelheden), Verwijst naar de grote hoeveelheden data die door verschillende systemen geproduceerd kunnen worden,
- **Velocity** (snelheid), Verwijst naar de snelheid waarmee data geproduceerd en verplaatst worden en geanalyseerd.
- **Variety** (veelvormigheid), Was vroeger data met name gestructureerd (relationele databases) tegenwoordig kunnen we vele bronnen gebruiken met ieder een eigen "structuur".
- **Veracity** (waarheidsgetrouwheid), Heeft betrekking op de kwaliteit van de verschillende data. Met name minder gestructureerde data kent vele verschijningsvormen met een wisselende kwaliteit.
- **Value** (waarde en ruis), Zijn de big datasets om te zetten in waarde voor de organisatie. Hebben de datasets voldoende "informatiegehalte" en staan de kosten van verwerking van deze data in verhouding met de informatie die eruit gehaald kan worden.

Gedistribueerde verwerking

Door de komst van big data ontstond er behoefte aan nieuwe vormen van dataverwerking. Met name data parallelisme wordt bij Big Data een essentieel onderdeel van de data verwerkingsarchitecturen. Een aantal redenen waarom nieuwe algoritmen noodzakelijk worden zijn hieronder nader toegelicht.

De dataset die geanalyseerd moet worden is dusdanig groot dat het verwerken of analyseren van de inhoud van deze data een te lange doorlooptijd kent, denk hierbij aan vele uren tot zelfs meerdere dagen. Daarnaast kan de snelheid waarmee de data het platform instroomt van deze data zo groot zijn dat daarmee de data ontvangst vastloopt. Als laatste is ook de veelvormigheid te noemen, de verwerking van veelvormige data transformeren naar een analyseerbare vorm of in een gewenst formaat is een vereiste. Hierbij zie je de 3Vs terug als kenmerk waarom gedistribueerde verwerking noodzakelijk is. Samenvattend kan gesteld worden: **Eén verwerkingseenheid kan de data verwerking niet aan binnen de gestelde tijd.** Dat was een reden om te zoeken naar andere vormen van dataverwerking.

Er zijn twee mogelijkheden

- **Scale up**, Een grotere verwerkingseenheid als de verwerkingseenheid onvoldoende in staat is kun je een verwerkingseenheid aanschaffen die de gestelde performance eisen wel kan invullen. Dit is natuurlijk een eindig pad op een gegeven moment heb je de snelst mogelijke verwerkingseenheid en is verdere scale up onmogelijk.
- **Scale out,** in plaats van een verwerkingseenheid ga je meerdere verwerkingseenheden inzetten die parallel de data gaan verwerken. De data wordt hiermee gelijktijdig verwerkt door een aantal verwerkingseenheden. Voordeel van scale out is dat je indien nodig steeds meer verwerkingseenheden kunt bijschakelen.

Het scale out patroon wordt gebruikt bij Big data voor gedistribueerde dataverwerking. Map Reduce is daarbij één van de eerste algoritmen die beschreven is. Vandaar dat we hier Map Reduce beschrijven.

Map Reduce

Map Reduce heeft de volgende kenmerken:

- Map Reduce wordt gebruikt in gedistribueerde data omgevingen waarbij de traditionele (sequentiële) verwerkingsalgoritmen niet mogelijk zijn.
- Map Reduce is gericht op data parallelisme. Dit houdt in dat een grote dataset wordt gesplitst en wordt opgeslagen op meerdere verwerkingseenheden. Deze verwerkingseenheden worden in een cluster geplaatst zodat de data gecoördineerd parallel door de verwerkingseenheden verwerkt kunnen worden.
- In Map Reduce worden de taken verdeeld over een Map en Reduce verwerkingstaken. Wat dit betekent wordt hieronder beschreven.
- Deze taken zijn onafhankelijk van elkaar en kunnen daarom op verschillende instances werken. Dit betekent dat in het cluster een aantal verschillende taken wordt toegewezen aan de verwerkingseenheden in het cluster.

Map Reduce en gedistribueerde dataverwerking zijn niet altijd toepasbaar. In sommige situaties is data parallelisme namelijk niet inzetbaar. Er worden daarom een aantal eisen gesteld aan het data verwerkingsalgoritme en/of de inhoud van de dataset.

- Bij relatief simpele algoritmen waarbij dezelfde logica op meerdere delen van de dataset toegepast wordt. Dat betekent dat de data is gesplitst over meerdere verwerkingseenheden waarbij het algoritme dat uitgevoerd dient te worden geschikt is voor de data zoals opgeslagen in de splits.
- Dataset is gedistribueerd opgeslagen, dus opgeslagen op meerdere verwerkingseenheden.
- De structuur van de dataset is bekend en het toe te passen algoritme is relevant voor de structuur van de dataset. Dat kan zijn dat de data gestructureerd is of dat het algoritme toe te passen is op ongestructureerde data. Bijvoorbeeld het tellen van woorden in een dataset is een algoritme is toe te passen op ongestructureerde data. De enige structuur op de data is dat inzichtelijk moet zijn wat een woord is. Bijvoorbeeld gescheiden door een spatie.
- De logica in het algoritme is op te delen in een Map en een Reduce functie. De Map functie in het algoritme wordt naar de splits op de verschillende verwerkingseenheden gestuurd om daar een tussenresultaat te bepalen van de verwerking van de data in de gesplitste data. Vervolgens is er een eindstap die de tussen resultaten verzameld en vervolgens het finale eindresultaat bepaald.
- Bij Map Reduce wordt een key value pair gebruikt dat zorgt voor de uitwisseling van tussenresultaten tussen de stappen in het Map Reduce algoritme.

MAP REDUCE STAPPEN

Het Map Reduce algooritme bestaat uit een aantal stappen. Map en Reduce komen altijd voor. De andere stappen zoals Combine en Shuffle en Sort komen alleen bij de complexere vormen van dataverwerking voor. Hieronder worden de stappen kort toegelicht.

- **Map,** Verdeling van de dataset in meerdere splits (split is een key value pair). Splits worden afzonderlijk naar de Mapper functie gestuurd. Mapper functie is de logica of het toe te passen algoritme. Uitvoer is nul, één of meerdere key value pairs.
- **Combine**, Dit is een optionele bewerking die niet bij elke MapReduce nodig is. Vooral bij grote datasets kan het interessant zijn om resultaten te combineren zodat er minder netwerkverkeer (naar reduce) (traag) nodig is. Combine functie aggregeert de resultaten van een Map. Het is daarmee een soort lokale reduce functie.
- **Partition**, wordt gebruikt als er meerdere reducers zijn en is daarmee optioneel. Partition verdeeld de uitkomst van meerdere map functies naar de relevante reducer functie. Partition kan ingezet worden om de workload van reducers gelijk te verdelen (ongelijke verdeling is vertraging).
- **Shuffle en Sort**, Uitvoer van alle partitioners is verdeeld over de reducer taken. Bijvoorbeeld voor het ontdubbelen van resultaten als dat nodig is. Vervolgens worden de elementen gesorteerd zodat er een groepering ontstaat. Sortering is configureerbaar. Shuffle en sort function brengt dan het resultaat terug tot één key value pair per groep.
- **Reduce,** Laatste stap in de verwerking dit kan een verdere groepering zijn maar ook ongewijzigd doorsturen van het resultaat. Verwerking is vanzelfsprekend wederom een configureerbaar algoritme. Houdt er rekening mee dat de reducer afhankelijk van het algoritme ook een leeg resultaat kan opleveren (filter). Als er meerdere reducers zijn worden de resultaten gecombineerd (in een laatste reducer).

MAP REDUCE VOORBEELD

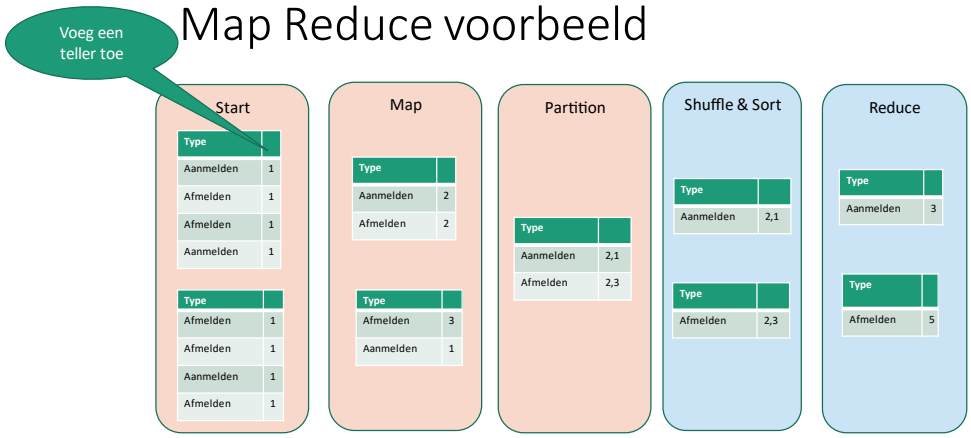
We hebben de stappen in het algoritme beschreven maar mogelijk is het nog steeds ietwat abstract. Laten we daarom een voorbeeld uitwerken waarbij het Map Reduce gevisualiseerd wordt op basis van twee splits. De tabel bestaat uit 8 rijen maar hou in gedachten dat het in werkelijkheid 8000.000.000 rijen zijn. Dan wordt gedistribueerde verwerking relevant.

Het voorbeeld bestaat uit een viertal kenmerken waarbij we voor ons algoritme er slechts één gebruiken, namelijk de data in de dataset binnen de kolom Type. We willen daarbij het aantal verschillende typen gaan tellen. Hieronder de tabel en daaronder een visualisatie van het algoritme binnen de stappen.

Verzender	Type	Datum	Status
1000	Aanmelden	14-11-2015	Ok
1200	Afmelden	14-11-2015	Ok
1000	Afmelden	14-11-2015	Ok
1400	Aanmelden	14-11-2015	Error
1200	Afmelden	14-11-2015	Ok
1000	Afmelden	14-11-2015	Ok

1400	Aanmelden	14-11-2015	Ok
1000	Afmelden	14-11-2015	Ok

Uitwerking



In de afbeelding zie je dat er een splitst gemaakt wordt waarbij de data gesplitst wordt naar twee verwerkingseenheden die beiden onafhankelijk van elkaar het tel algoritme toepassen op de data. Daartoe wordt er binnen het algoritme een teller toegevoegd. Vervolgens berekent iedere mapper het eindresultaat van de data in de eigen split en geeft dit eindresultaat door in de vorm van een key value paar.

Vervolgens gaat het via een aantal tussenstappen uiteindelijk naar de reducer die het finale eindresultaat gaat bepalen op basis van de key value pairs en berekent vervolgens de finale key value pairs als eindresultaat.

Map Reduce is daarmee een eenvoudig toe te passen algoritme is. Ondertussen zijn er op basis hiervan al vele opvolgers ontstaan. Wat betreft de opvolgers van MapReduce, deze zijn ontwikkeld om snellere verwerking, betere schaalbaarheid en efficiënter gebruik van middelen te bieden. Technologieën zoals Apache Hadoop's YARN, Apache Spark en cloudservices zoals Amazon EMR en Google Cloud Dataproc zijn enkele van de bekende opvolgers. Deze systemen bieden verbeteringen zoals in-memory processing, real-time stream dataverwerking en geavanceerde data-analysefuncties.

NOSQL DATABASES

Inleiding

In de voorgaande hoofdstukken zijn we uitgebreid ingegaan op databases en met name relationele databases. Zoals reeds aangegeven zijn relationele databases reeds vele tientallen jaren aanwezig. Met name ter ondersteuning van administratieve systemen maar ook voor data analyse en de ondersteuning van kenniswerkers worden relationele databases veelvuldig ingezet.

Ondertussen zijn er naast relationele databases ook een aantal nieuwe database technologieën ontstaan die ondersteuning beiden aan een aantal andere concepten. Met name op het vlak van nieuwe data structuren. Deze nieuwe concepten worden allemaal onder de noemer NoSQL gebracht.

De term “NoSQL” werd voor het eerst gebruikt door Carlo Strozzi in 1998 om zijn lichtgewicht opensourcedatabase te benoemen. Deze database bood geen SQL-interface aan, maar was nog steeds relationeel. Het is belangrijk op te merken dat zijn NoSQL RDBMS verschilt van het algemene concept van NoSQL-databases dat rond 2009 ontstond. Kortom, NoSQL-databases zijn een breed scala aan databasemanagementsystemen die aanzienlijk verschillen van de klassieke relationele databases. Ze hebben geen vaste databaseschema’s nodig, vermijden meestal JOIN-operaties en schalen vaak horizontaal. Horizontaal schalen is in vorige hoofdstuk beschreven onder scale out.

VOORBEELDEN NOSQL TOEPASSINGEN

Natuurlijk! NoSQL-databases worden vaak gebruikt voor toepassingen met extreem grote datasets. Hier zijn enkele voorbeelden [Copilot]:

- **Social Media Platforms**: Bedrijven zoals Twitter, Facebook en Google verzamelen dagelijks terabytes aan gebruikersgegevens. NoSQL-databases helpen hen bij het beheren van deze enorme hoeveelheid gegevens, zoals gebruikersprofielen, berichten, likes en volgers.
- **E-commerce Toepassingen**: NoSQL-databases zijn ideaal voor e-commercebedrijven die grote hoeveelheden gegevens moeten verwerken, zoals productcatalogi, klantprofielen en transactiegeschiedenis.
- **Realtime Analyse**: NoSQL-databases worden gebruikt voor realtime analyse van gegevensstromen. Bijvoorbeeld, het bijhouden van gebruikersactiviteit op een website of app, het monitoren van sensordata in Internet of Things (IoT)-toepassingen, en het detecteren van fraude in financiële transacties.
- **Aanbevelingssystemen**: NoSQL-databases worden ingezet voor aanbevelingssystemen, waarbij verbindingen worden gelegd tussen mensen op basis van verschillende soorten gegevens. Denk aan Netflix die films aanbeveelt op basis van kijkgeschiedenis of Amazon die productaanbevelingen doet op basis van aankopen.
- **Grafen**: NoSQL-databases zoals Neo4j worden gebruikt voor het opslaan en bevragen van complexe relaties tussen entiteiten, zoals sociale netwerken, organisatiehiërarchieën en kennisgrafieken.

- **Tijdreeksen**: Toepassingen die enorme hoeveelheden tijdreeksgegevens genereren, zoals logboeken, sensordata en financiële marktgegevens, profiteren van NoSQL-databases zoals InfluxDB of Cassandra.

In dit hoofdstuk gaan we de meest voorkomende NoSQL database typen kort bespreken.

Karakteristieken NoSQL

In dit hoofdstuk gaan we in op een aantal karakteristieken van databases om de verschillen tussen de verschillende NoSQL databases toe te lichten.

KENMERKEN NOSQL

NoSQL onderscheidt zich ten opzichte van relationele databases door een aantal kenmerken:

- **Schemaloze of schema-arme opslag**, relationele databases kenmerken zich door de structuur en de functionaliteiten aanwezig om zorg te dragen dat de data in de database aan de eisen binnen de beschreven structuur voldoet. Bij NoSQL databases is dat duidelijk niet het geval. In een aantal gevallen worden er helemaal geen eisen gesteld aan de data en is validatie van deze data door de database niet mogelijk (er is geen structuur). In andere gevallen is er wel enige structuur maar ook hierbij kan hiervan afgeweken worden. Dit wordt vooral toegepast bij semi gestructureerde data.

- **Schema on read**, Is een relationele database gebaseerd op schema on write, bij het schrijven wordt de data gevalideerd. Bij NoSQL dat is het paradigma het tegenovergesteld. Bij het schrijven van de data naar de database vindt er geen, of beperkte, validatie plaats. Dit wordt op een later stadium gedaan. Als de data in een later stadium gebruikt gaat worden voor data analyse of –verwerking wordt het datamodel en de regels toegepast in de data bepaald en wordt de data ingezet binnen data gedreven toepassing .

- **Scale out ipv scale up**, in het vorige hoofdstuk zijn we ingegaan op gedistribueerde verwerking van data om hiermee schaalbaar te zijn voor zeer grote datasets. Scale out is voor NoSQL een gangbaar concept waarmee gedistribueerde verwerking ter ondersteuning van extreem grote datasets mogelijk is.

- **Hoge performance bij schrijven**, schrijven van data naar een relationele database is relatief traag. Dit komt door de transformatie die uitgevoerd moet worden om de data die naar een database geschreven wordt aan het schema te laten voldoen. Daarnaast gaat de relationele database een aantal controles uitvoeren op de data bij het schrijven.
  Bij NoSQL databases hebben we gezien dat het toegepaste concept “schema on read” is dus bij het schrijven worden er geen validaties uitgevoerd. Daarom dus geen transformaties voorafgaand aan het schrijven en ook geen validaties. Daarnaast wordt ook voor het wegschrijven van de data gedistribueerde verwerking toegepast Het gevolg is een enorme performance verbetering ten opzichte van relationele databases.

- **Auto sharding en replicatie**, Bij NoSQL databases wordt veelal auto sharding en replicatie toegepast. Dat betekent dat de data reeds bij het schrijven naar de database in splits wordt opgeslagen. Vervolgens wordt de data gerepliceerd naar een aantal extra nodes in het cluster.

Hiermee wordt optimaal gebruik gemaakt van de mogelijkheden die een cluster van nodes voor dataopslag biedt. Belangrijk hierbij is de term Auto wat inhoud dat de sharding en replicatie configureerbaar is in een database en dat dit vervolgens geautomatiseerd wordt uitgevoerd. Gevolg is dat er een aantal extra kenmerken mogelijk worden zoals hoge beschikbaarheid.

- **Hoge beschikbaarheid**, door het toepassen van auto sharding replicatie wordt data gedistribueerd en met een aantal kopieën opgeslagen in het cluster. Dat levert een hoge beschikbaarheid om. Is een node in het cluster niet beschikbaar dan zal de data nog op één of meerdere andere nodes staan waarnaar de dataverwerking ingezet kan worden.

- **Niet volledig ACID compliant**, relationele databases zijn ACID compliant wat inhoud dat een bewerking van de data volledig lukt of mislukt. Als de bewerking mislukt draagt de database er zorg voor dat de database teruggebracht wordt in de staat waarin de data verkeerde voordat de bewerking gestart werd.
  NoSQL databases zijn niet of niet volledig ACID compliant. Dat is voor transactionele dataverwerking een nadeel. Echter een aantal NoSQL database typen benaderen de ACID compliant werkwijze steeds meer. Daarnaast is voor een aantal soorten dataverwerking ACID compliancy minder van belang.

- Laag in kosten en te installeren op eenvoudige hardware, omdat NoSQL gebaseerd is op gedistribueerde verwerken waarbij gewerkt wordt met Auto Sharding en Replicatie kan gekozen worden voor goedkopere hardware. Door het werken met een cluster waarin de database gedistribueerd is opgeslagen kan een node uitvallen en wordt deze overgenomen door een andere node in het cluster. Hiermee hoeven minder hoge eisen gesteld te worden aan de hardware.
  Daarnaast zijn vrijwel alle NoSQL databases gebaseerd op Open Source licentiemodellen wat een positief effect heeft op lage licentiekosten. Houd er wel rekening mee dat NoSQL wel een investering vraagt in het opdoen van kennis en mogelijk ondersteuning vanuit het perspectief van service management.

DATA STRUCTUREN

Relationele databases kenmerken zich dat data is opgeslagen in tabellen en dat de data in verschillende tabellen door middel van joins aan elkaar gerelateerd kunnen worden. De tabel als datastructuur is kenmerkend voor het werken met data maar kent ook een aantal beperkingen.

Binnen NoSQL zijn een aantal andere datastructuren geïmplementeerd waardoor ook data die niet de structuur heeft van een tabel kan worden opgeslagen. Dat kan afhankelijk van de data gedreven toepassing een voordeel zijn. In NoSQL worden een drietal datastructuren ingezet naast de tabel datastructuur. Hieronder worden de datastructuren kort toegelicht. Later in dit hoofdstuk worden ze bij de NoSQL databasetypen verder beschreven.

- **Column family datastructuur**: dit is een verzameling van rijen, waarbij elke rij een willekeurig aantal kolommen kan bevatten. Dus de data is wel opgeslagen in een tabelvorm maar elke rij kan een subset van de gedefinieerde kolommen bevatten. Alle kolommen in de column family is

daarmee de superset van de kolommen die een rij kan bevatten. Daarnaast is in een later stadium het datamodel van een column family eenvoudig uit te breiden.

- Document datastructuur: Het belangrijkste verschil van een document database ten opzichte van relationele database is dat de data is opgeslagen in een boomstructuur. Alles bestaat uit sleutel en waarde paren waarbij het waarde element weer uit sleutel waarde paren bestaat waardoor een boomstructuur ontstaat. Documenten kunnen verschillende soorten en structuren bevatten, zoals strings, getallen, datums, arrays of objecten. Ze worden vaak opgeslagen in formaten zoals JSON, BSON of XML.
  Hier is een voorbeeld van een JSON-document in een document database:

```
{
  "_id": 1,
  "first_name": "Tom",
  "email": "tom@example.com",
  "cell": "765-555-5555",
  "likes": ["fashion", "spas", "shopping"],
  "businesses": [
    {
      "name": "Entertainment 1080",
      "partner": "Jean",
      "status": "Bankrupt",
      "date_founded": { "$date": "2012-05-19T04:00:00Z" }
    },
    {
      "name": "Swag for Tweens",
      "date_founded": { "$date": "2012-11-01T04:00:00Z" }
    }
  ]
}
```

- Grafen datastructuur, het grafen datamodel, is gebaseerd op knooppunten (nodes) en verbindingen (edges).
  Nodes zijn de basisentiteiten in een grafiek. Ze vertegenwoordigen objecten, entiteiten of concepten. Elke edge kan eigenschappen bevatten. Deze eigenschappen zijn sleutel-waardeparen die specifieke

informatie over het knooppunt bevatten. Bijvoorbeeld, als we een sociale netwerkgrafiek hebben, kunnen nodes gebruikers, berichten, pagina's, enz. voorstellen. Elk knooppunt heeft eigenschappen zoals naam, leeftijd, locatie, enz.

Edges (ook wel relaties of bogen genoemd) verbinden nodes in een grafiek. Edges hebben ook eigenschappen. Deze eigenschappen kunnen extra informatie bevatten over de relatie tussen de gekoppelde nodes. Bijvoorbeeld, in een vriendennetwerkgrafiek, kan een edge "vriend van" zijn en eigenschappen bevatten zoals "sinds wanneer" of "sterkte van de vriendschap".

Er zijn meerdere soorten Grafen:

- **Gerichte grafen**: Hierbij hebben edges een richting. Ze gaan van het ene knooppunt naar het andere.
- **Ongerichte grafen**: Hierbij zijn edges niet-gericht. Ze verbinden twee knooppunten zonder een specifieke richting.
- **Gewogen grafen**: Edges hebben een gewicht of waarde die de sterkte van de relatie aangeeft.
- **Multigrafen**: Hierbij kunnen meerdere edges tussen dezelfde knooppunten bestaan.

[Bron: Copilot]

NoSQL database soorten

Rond de verschillende datastructuren die ingezet worden zijn in de afgelopen jaren naast Relationele databases nieuwe soorten databases ontstaan. Dat zijn enerzijds de NoSQL databases maar ook een gedistribueerd bestandsysteem mag hierbij niet ontbreken. In dit hoofdstuk gaan we de karakteristieken op basis van voor en nadelen met elkaar vergelijken. Daarom werken we relationele databases daarom ook uit in deze paragrafen

- Relationele databases
- Gedistribueerd bestandssysteem
- Key-value store
- Document database
- Column-family
- Graph database

Elke paragraaf kent daarbij een zelfde opbouw namelijk

- Opsomming van kenmerken met een toelichting
- Visualisatie datastructuur
- Voor en nadelen

RELATIONELE DATABASE

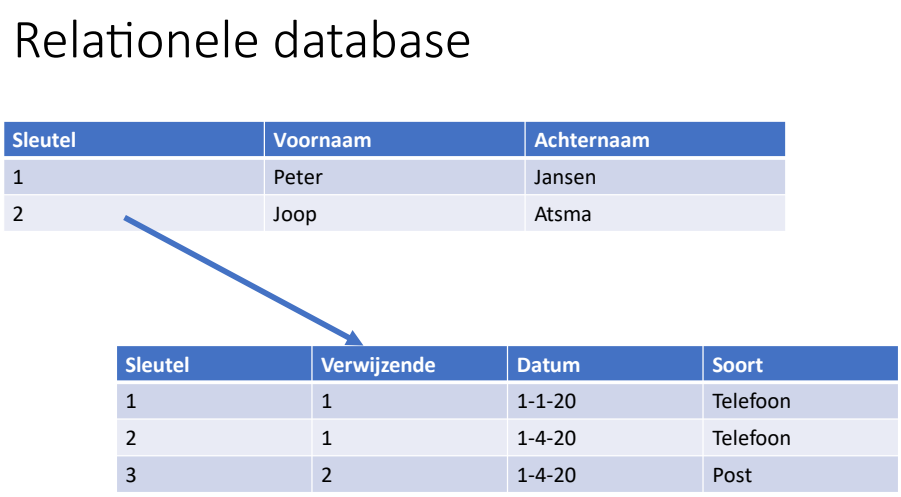
De relationele database is reeds uitvoerig besproken. Hier daarom een korte samenvatting met de kenmerken en voor- en nadelen.

Kenmerken

- Een relationele database is vooral geschikt voor de opslag van transactionele data. Dit vanwege de structuur in een relationele database en de functionaliteiten van data validatie op basis van de structuur.
- RDBMS is **ACID compliant, een bewerking op de data lukt of mislukt in zijn geheel en voor en na de bewerking is de data valide op basis van de structuur**
- Meestal geïnstalleerd op een single node, partitioneren van de data over meerdere nodes is niet mogelijk vanwege de validaties op basis van de structuur van de data in relatie tot de reeds aanwezige data in de database.
- Scale up i.p.v. scale out, vanwege het single node aspect en het niet kunnen partitioneren van de data maakt dat scale out niet mogelijk is
- RDBMS slaat data veelal op in een (tabel) schema, structuur is gebaseerd op tabellen die op basis van extra validatie regels aan elkaar gerelateerd kunnen worden
- Muteren van data is daarmee relatief traag, vanwege de validatie op consistentie maar ook de transformatie naar een tabelvormige datastructuur is muteren en toevoegen van data relatief traag.

Datastructuur

In de afbeelding hieronder zie je dat de data binnen een database in een aantal tabellen wordt opgeslagen. Daarnaast kunnen de tabellen op basis van sleutels aan elkaar gerelateerd worden. Dit worden joins genoemd.



Evaluatie relationele database

Positieve kenmerken

- ACID compliant op basis van rollback (terugdraaien) en commit (bevestigen) van transacties
- Verwerken van transactionele data op basis van de tabelstructuur. De tabelstructuur is daarbij bij voorkeur genormaliseerd wat inhoud dat duplicaten van attributen beperkt worden in de data.
- Consistency en validatie op basis van schema
- Querytaal is gestandaardiseerd op basis van SQL wat een gestructureerde en gestandaardiseerde querytaal is

Negatieve kenmerken

- Meestal single node in verband met ACID en validatie van de data in de tabel
- Relatief trage mutaties en toevoegingen van data in de tabellen
- Binaire en semigestructureerde opslag minder ondersteund, met name hoog structurele data ondersteund
- Relatief duur in verband met licentiekosten en beheerkosten. Beheren van relationele databases vraagt specifieke expertise en wordt meestal uitgevoerd door een specialistische medewerker.

GEDISTRIBUEERD BESTANDSYSTEEM SYSTEEM

Het gedistribueerde bestandsysteem is veel breder inzetbaar dan een database. Het is vergelijkbaar met een bestandsysteem zoals Windows of Linux maar heeft daarbij een bijzondere eigenschap namelijk: gedistribueerde opslag van de files over meerdere nodes in een cluster.

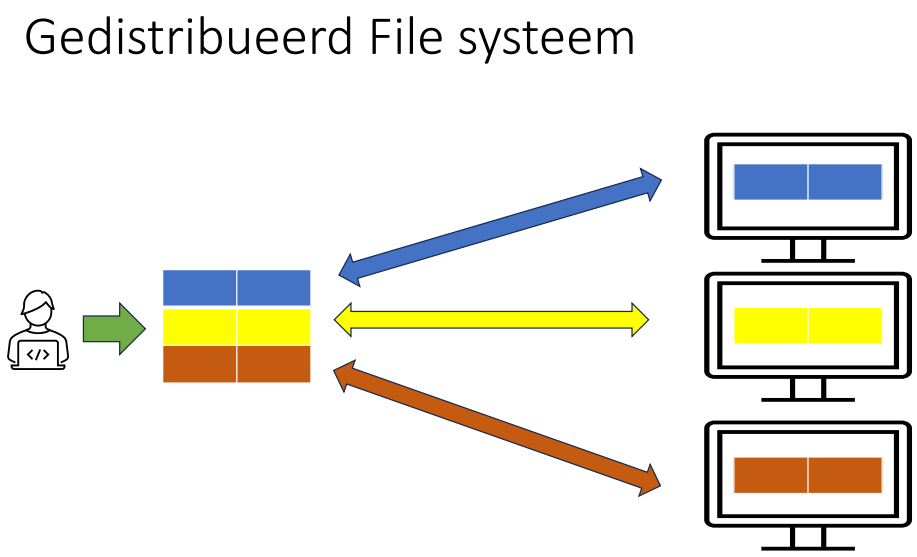
Daardoor ontstaan er in een gedistribueerd bestandsysteem om gebruik te maken van algoritmen zoals Map Reduce om de performance, schaalbaarheid en fouttolerantie te verbeteren.

Kenmerken

- Opslag van grote hoeveelheden data verdeeld over meerdere filesystemen, daarmee zeer schaalbaar. Er is een cluster bekend van 2000 nodes in het cluster. Zie https://www.skillvertex.com/blog/who-has-the-worlds-largest-hadoop-cluster/
- Simpele en snelle toegang tot datasets in files zonder relationele verbindingen. Het bestandssysteem is net zo simpel in het gebruik zoals een Windows verkenner. Achter de schermen gaat een complexe inrichting van gedistribueerde inrichting van de files plaats.
- Vooral geschikt voor data waarbij streaming toegang nodig is, streaming van data kenmerkt zich dat het toevoegen van nieuwe data eenvoudig gaat maar achteraf muteren van de data is minder gebruikelijk.
- Minder geschikt voor een dataset bestaande uit veel kleine bestanden, veroorzaakt door de achterliggende registratie waarin bijgehouden dient te worden waar de partities en replica's van de data zich bevinden op het cluster van nodes.

Datastructuur

Zoals reeds genoemd ziet het bestandsysteem eruit als een aantal files. Achter de schermen worden de bestanden opgesplitst in partities en die worden verspreid over de verschillende nodes. Ook wordt replica toegepast waarbij er kopieën gemaakt van deze partities. De registratie en administratie die daarbij hoort bevindt zich in de software van het HDFS (Hadoop Distributed File System) en is voor de eindgebruiker onzichtbaar.



Evaluatie gedistribueerd bestandsysteem

Positieve kenmerken

- Gedistribueerde data opslag en daarmee ook gedistribueerde dataverwerking en -analyse.
- Implementatie van Map Reduce en opvolgers, dit zijn algoritmen die de data gedistribueerd opslaan en vervolgens gedistribueerd verwerken. Daarmee wordt een enorme performance winst gehaald.
- Bestand gebaseerd, in veel toepassingen is het werken met data in bestanden relatief eenvoudig maar goed inzetbaar.
- Meerdere componenten in oplossing specifieke doeleinden, in bijvoorbeeld Hadoop (https://hadoop.apache.org/) is een ecosysteem te vinden van vele componenten en libraries die allemaal eigen functionaliteiten bieden voor de bewerking van data in bestanden.

Negatieve kenmerken

- Veel kleine bestanden minder ondersteuning, heeft te maken met de registratie en het beheer van de partities en replica's van deze partities binnen het filesysteem.
- Security inrichting is complex omdat de data wordt opgeslagen op meerdere bestandsystemen inclusief de replica's.
- Met name batch georiënteerd. Hadoop is zeer geschikt voor batchverwerking. Het verdeelt taken in mappers en reducers, waardoor het efficiënt grote hoeveelheden gegevens kan verwerken. Er zijn wel een aantal componenten in het ecosysteem voor streaming dataverwerking.

KEY VALUE STORE

De key value store is qua data structuur het meest eenvoudig. Het doet namelijk niets met de structuur van de data. Ik gebruik vaak de metafoor met een locker of bagagekluis. Het maakt niet uit wat je erin stopt alle soorten data zijn toegestaan. Alleen als je de sleutel hebt komt de data er weer uit. Dat is een korte samenvatting van de key value store.

Kenmerken

- Eenvoudige opslag van een sleutel en een bijbehorende waarde. Als de data erin gestopt wordt krijg je een gegenereerde sleutel. De data kan alleen worden opgehaald als je de sleutel hebt . Sleutels zijn de enige manier om een element op te zoeken.
- De data kan van alles zijn variërend van sensor data tot een video, foto etc.
- Gedeeltelijke updates van data is niet mogelijk want voor de database is het slechts een BLOB aan data zonder enige structuur.
- Sommige gebruiken extra concepten als buckets of collections voor extra indelingsmogelijkheden. Bijvoorbeeld inkomende en uitgaande buckets of andere indelingen in de tijd op basis van de buckets of collecties.

Datastructuur

De structuur van de data in een key value store is eenvoudig het is een blok van binaire data en zo wordt het ook behandeld door de database.

Key Value structuur

Key	Value
1000	{ sender: 1200 }, { Type: afmelden}
1500	00010111011011010101000010001011111100010110010 1
1209	Voorbeelden van key value stores zijn Riak, Redis en DynamoDB

Evaluatie Key Value store

Positieve kenmerken

- Opslag van ongestructureerde data, de database doet niets met de data
- Hoge verwerkingssnelheid bij schrijven, geen verwerking, geen validatie geen transformatie. Daarnaast gedistribueerde verwerking (ook bij het schrijven) voor maximale performance.
- Sleutel identificatie is voldoende, sleutel wordt gegenereerd en is de enige manier om later toegang te krijgen tot de data.
- Opgeslagen waarden worden bewerkt via de applicatie, de structuur van de data is binnen de database onbekend dus dit zal in een applicatie moeten gebeuren die de structuur wel kan bepalen.

Negatieve kenmerken

- Geen zoeken en filteren in data, structuur van de data is onbekend dus zoeken en filteren is zinloos.
- Geen relaties tussen de elementen, structuur van de data is onbekend dus relatie zijn afwezig in de data.
- Groepsmutaties op elementen en/of attributen niet mogelijk, structuur van de data is onbekend dus mutaties zijn onmogelijk.
- Geen schema consistentie, structuur van de data is onbekend dus validaties zijn niet mogelijk.

COLUMN FAMILY

Column family is een database met een tabel georiënteerde dataopslag. Hierbij wordt gewerkt met column families. Iedere column family kan een eigen set aan kolomdefinities hebben. Echter iedere rij in een column family kan een subset van deze kolommen vullen met data naar behoefte van de registratie van dat record.

Kenmerken

- Opslag lijkt veel op een relationele database, tabelgeorienteerde aanpak maar met een aantal vrijheden in het vullen van de data in de rijen binnen een table family.
- Heeft zogenaamde column families, dit is een groepering van kolommen die logischerwijs bij elkaar horen. Maakt het dus mogelijk om kolommen te classificeren op basis van een family.
- Een kolom kan bestaan uit een aantal gerelateerde kolommen (super column)
- Snelle CRUD operaties met random read write, vanwege de gestructureerde opzet en de opbouw van de datastructuur is het doen van lezen en mutaties zeer snel. Mede vanwege de gedistribueerde dataopslag.
- Ondersteuning van flexibele schema's (rijen met een eigen structuur obv column families e.d).

Datastructuur

De structuur bestaat uit column families. In het voorbeeld in de afbeelding de families Ontvanger, Verzender en Bericht. Daarbinnen zijn een aantal kolommen gedefinieerd. Binnen een rij wordt de data naar een cel voor een kolom weggeschreven op basis van een sleutel waarde paar. Een cel kan ontbreken

binnen een rij terwijl die wel in de column family wel gedefinieerd is. Dat maakt het een flexibel database schema.

Structuur column family

Id	Ontvanger	Verzender	Bericht
1000	{ site: AMD identifier: 965 contact: Peter }	{ site: PTR identifier: 327 contact: John phone: 12345678 }	{ identifier: 9645 Type: Afmelden subject: 964567 subject_name: Kok }
1245	{ site: OTL identifier: 835 contact: Paul }	{ site: AMD identifier: 965 contact: Peter }	{ identifier: 9645 Tijd: Afmelden subject: 964567}

Evaluatie Column Family

Positieve kenmerken

- Realtime random readwrite mogelijk, vanwege de gestructureerde opzet en de werkwijze met sleutel waarde paren
- Data heeft een tabelstructuur, is voor data analyse en transactionele data vaak een voordeel, omdat dat ook een tabelstructuur kent.
- Schema evolutie wordt ondersteund, uitbreiden van een column family met nieuwe kolommen is goed mogelijk.
- Logische groepering van velden voor zoeken, column families zijn feitelijk een soort classificatie en daarmee een groepering van kolommen.
- Efficiënte gegevensopslag, door de sleutel waarde paren efficiënte werkwijze. Daarnaast is er geen ruimte nodig als een rij geen waarde heeft voor data behorend bij een kolom.

Negatieve kenmerken

- Niet ACID, Hoewel sommige column family databases enkele ACID-eigenschappen ondersteunen, zijn ze over het algemeen niet volledig ACID-compliant. Dit komt door hun focus op schaalbaarheid en prestaties.
- Geen joins mogelijk, In een column family moedigt de database denormalisatie aan. Dit betekent dat je gegevens moet dupliceren en structureren op basis van de query's die je later wilt uitvoeren. Je kunt meerdere kopieën van gegevens opslaan om efficiënte query's mogelijk te maken zonder joins. Er zijn geen **foreign keys**: De database heeft geen concept van foreign keys

zoals relationele databases. Er is geen ingebouwde manier om gegevens uit meerdere tabellen te combineren.

- Geen SQL compliant queries, sommige column family databases hebben een eigen query taal bijvoorbeeld CassandraQL andere databases werken met Put, Post en Get paradigma's om data te lezen en schrijven.
- Opslag van binaire data minder geschikt. Hoewel blobs handig zijn voor specifieke use-cases, moet je voorzichtig zijn met de grootte van de cellen. Grote cellen kunnen de prestaties van een column family database negatief beïnvloeden.
- Schema mutatie minder ondersteund, uitbreiden is goed mogelijk muteren van de datastructuur is door de opbouw van sleutel waarde paren minder goed mogelijk.

DOCUMENT DATABASE

De document database slaat data structuren op in de vorm van een boomstructuur. Daarbij is een document opgebouwd uit sleutel waarde paren. Waarbij de waarde weer uit sleutel waarde paren kan bestaan. Hierdoor ontstaat op natuurlijke wijze een boomstructuur. Kenmerkend daarbij is dat de sleutel feitelijk metadata is dat de inhoud van de waarde daarmee beschrijft. Echter in een document database kunnen sleutel waarde paren ontbreken.

De data in een document database is semi gestructureerd en zelf beschrijvend. Het heeft een nauwe relatie met XML en JSON wat eenzelfde datastructuur heeft. Is er in een data gedreven toepassing veel XML of JSON aanwezig dan is een document database een logische keuze. Je kunt feitelijk de data zonder transformatie van een XML of JSON bericht rechtstreeks naar de database schrijven zonder transformatie.

Kenmerken

- Opslag in een sleutel waarde maar de waarde heeft zelf een complexe geneste- of boomstructuur
- De opgeslagen structuur is zelf beschrijvend, omdat de sleutel de metadata is en daarmee context geeft aan de waarde binnen het paar.
- Mogelijk om te zoeken in deze structuur op basis van attributen. Door gebruik te maken van de sleutel waarde paren en de metadata kan een datamodel gegenereerd worden op basis van de inhoud van het document waarmee je vervolgens kunt zoeken in de boomstructuur.
- Mogelijk om een deel van de structuur via een select op te vragen, binnen de boomstructuur kun je een "tak" opvragen.
- Mogelijkheid van gedeeltelijke updates, binnen de boomstructuur kun je een "tak" vervangen met een andere "tak" wat weer een boomstructuur is van zichzelf.
- Indexeer mogelijkheden, omdat er een metadata aanwezig is die structuur biedt is het mogelijk om de performance te verbeteren op basis van index functionaliteit.

Datastructuur

In de afbeelding is te zien dat een document database uit een collectie van documenten bestaat en dat ieder document uit sleutel waarde paren bestaat maar dat ieder sleutel en waarde paar kan worden opgebouwd uit andere sleutel waarde paren. Een waarde in een sleutel waarde paren kan uit een tak bestaan die weer uit takken bestaat. Maar een tak kan ook leeg zijn en wordt dan weggelaten. Daarmee

ontstaat een zeer flexibele opbouw van een document op basis van het wel of niet vullen van een sleutel waarde paar.

Document database

document	document
{ id: 100, sender: 1200, Type: Error, date: 13-4-2012 error_Tijd: -99, componenten: [dbms: Oracle, erp:sap, office: word, crm: Dynamic, office: Prezi] }	{ sender: 1200, receiver: 1400, date: 12-4-2012, Type: Message, message_Type: Afmelden customers: [{ id: 1023, name: John } , { id: 1423, name: Pete , age: 16 }, { id: 6423, name: Carla, level: High }] }

Evaluatie document database

Positieve kenmerken

- Opslag van (geneste) semi structurele data met een nauwe relatie met XML en JSON.
- Zoeken op basis van verschillende meerdere attributen, omdat de sleutel het metamodel vormt is dit te gebruiken voor zoeken in de data.
- (gedeeltelijke) CRUD operaties, bewerkingen op de data zijn goed mogelijk, het bijwerken van takken door een tak te vervangen door een andere tak.
- Grotendeels ACID compliant, Door de opzet van documenten in een XML structuur wordt ACID compliant benaderd. Vanaf MongoDB 4.0 ondersteunt ACID-transacties met meerdere documenten, en versie 4.2 introduceerde gedistribueerde ACID-transacties met meerdere documenten voor nog meer flexibiliteit. MongoDB voldoet aan de ACID-verwachtingen voor betrouwbare datatransacties!

Negatieve kenmerken

- Meerrijige updates niet mogelijk, omdat ieder document een eigen structuur kan hebben is meerrijig updaten niet goed mogelijk.
- Beperkte mogelijkheid voor joins. In MongoDB kun je documenten uit verschillende collecties samenvoegen met behulp van de $lookup-stage. Deze stage stelt je in staat om aan te geven welke collectie je wilt samenvoegen met de huidige collectie en welke velden in elk document moeten overeenkomen, echter binnen één document is een query eenvoudiger van opbouw.
- Enig schema ontwerp is nodig, vanuit document database perspectief niet noodzakelijk maar voor later gebruik bij Schema on Read is enig schema ontwerp ten zeerste aan te raden.

- Beperkte opslag van binaire data. Standaard zijn MongoDB-objecten beperkt tot 16 MB in grootte. Als je grotere binaire gegevens wilt opslaan, worden deze “gechunkt” in meerdere objecten die elk kleiner zijn dan 255 KB.

GRAAF DATABASE

Bij de graaf database staan knooppunten (nodes) en verbindingen (edges) centraal. Grafen is een veelvuldig toegepaste datastructuur vooral voor data- en kennismodellen. Kennis heeft namelijk van zichzelf een graaf structuur. De graaf en de term graph kan voor ons als Nederlanders wat verwarrend zijn omdat een graaf voor ons meerdere definities heeft en in het Engels is dat ook het geval. Het heeft in ieder geval niets met grafieken te maken.

Het voert wat te ver om de grafenwiskunde in detail te gaan bespreken ook al is het volgens mij een super interessant onderwerp. Wil je meer informatie kijk dan eens op https://nl.wikipedia.org/wiki/Grafentheorie. Echter grafen worden wel in heel veel toepassingen toegepast, met name op social media platformen.

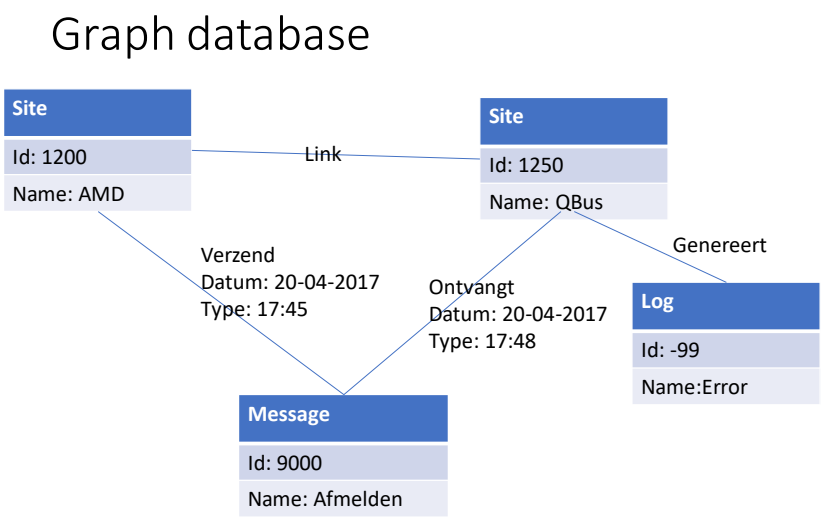
Kenmerken

- Opslag van verbonden data elementen (nodes en edges), er kunnen meerdere node en edge soorten zijn. Daarnaast kunnen er meerdere edges bestaan tussen twee nodes. Bijvoorbeeld twee personen die zowel collega, vriend als leidinggevende van elkaar zijn.
- Nodes en edge kan eigen attributen hebben, niet alleen nodes kunnen attributen hebben dat geldt ook voor de edges. Dat biedt krachtige mogelijkheden voor het bepalen van een datamodel.
- Edges kunnen ongericht-, uni- en bi directioneel zijn. Er zijn algemene extra kenmerken bij edge kenmerken. Bijvoorbeeld vader van, huisgenoot van, maar ook definiëren dat er een relatie is zonder kenmerken is mogelijk.

Datastructuur

De datastructuur van de data in een graaf database is bijzonder uitgebreid en biedt daarmee volop mogelijkheden om in detail een beschrijving te maken van de context die in een grafen database wordt opgeslagen. Met name de mogelijkheid om op edges attributen vast te leggen is een krachtig concept.

Grafen databases zijn goed in het modelleren van complexe relaties, zoals sociale netwerken, aanbevelingssystemen en kennisgrafieken. Grafen-databases gebruiken indexvrije toegang, wat betekent dat ze efficiënt relaties kunnen volgen zonder uitgebreide indexen. Grafen databases kunnen cyclische relaties detecteren, wat handig is bij het modelleren van complexe gegevens. Vaak gebruiken grafen databases een specifieke querytaal. Deze querytalen maken het bevragen van een database in veel gevallen veel eenvoudiger dan we gewend zijn bij relationele databases en SQL waar het uitschrijven van de joins tussen tabellen arbeidsintensief is.



Evaluatie grafen database

Positieve kenmerken

- Opslag van verbonden entiteiten met behulp van edges en nodes wat nauw aansluit bij bestaande toepassingen zoals het modelleren van kennis.
- Zoeken en filteren op basis van connecties, binnen een grafen database wordt een objectmodel opgesteld en dit object model vertaald zich naar gestructureerde data die vervolgens gebruikt kan worden voor het filteren en analyseren van de data op basis van de structuur.
- Mogelijkheid om te zoeken naar patronen. Door de querytaal is het zoeken naar patronen relatief eenvoudig taal. Daarnaast is door de structuur in de graaf het mogelijk om van graphql achtige talen gebruik te maken. Hierbij kan ook kunstmatige intelligentie worden ingezet.
- Grafen databases bieden de mogelijkheid om binaire date op te slaan door het datatype binary te gebruiken.

Negatieve kenmerken

- Muteren van grote groepen edges en nodes niet mogelijk, door de opzet van het model is het doen van groepsmutaties minder goed mogelijk.
- Grote aantallen attributen in een element werkt vertragend, het datamodel is minder flexibel dan bij een document database. Echter in een aantal gevallen kunnen graaf en document databases gecombineerd worden.
- Deels ACID, de data is gestructureerd op basis van datatype en op basis daarvan voert de database consistentie controles uit. Echter validaties op basis van primaire en verwijzende sleutels wordt niet door een graaf database gedaan.

Vergelijking databases

In de voorgaande paragrafen zijn we ingegaan op verschillende database typen die toegepast kunnen worden. Hieronder volgt een samenvatting om de databases op basis van de genoemde kenmerken in een tabel met elkaar te vergelijken.

Kenmerk	RDBMS	Gedistribueerd Bestandsysteem	Key Value store	Document database	Column family	Grafen
Datastructuur						
Gestructureerd	++	++	--	+	+	++
Semi gestructureerd	-	++	+	++	++	-
Ongestructureerd	--	++	++	+	+	-
Tabel	++	+	--	+	++	+
Boom	--	--	--	++	--	--
Graaf	--	--	--	--	--	++
Validatie						
Primaire sleutel	++	--	+	++	++	++
Verwijzende sleutel	++	--	--	--	--	--
Datatypen	++	--	--	++	+	++
Schema on read	--	++	--	+	++	+
Schema on write	++	-	--	+	+	++
ACID	++	--	--	-	+	++
Binaire data	-	++	++	-	+	+

Over de auteur

Bert Dingemans is trainer op het vlak van data architectuur, data management en Big Data. Hij heeft een passie voor modelleren, modelleertools en het effectief inzetten van geautomatiseerde hulpmiddelen om modellen effectief in te zetten in de praktijk. Meer whtepapers zijn te vinden op https://data-docent.nl. Bert is per mail te bereiken via bert@data-docent.nl