|  |  |
| --- | --- |
| ONDERZOEK DOCUMENT ORIENTED DATABASES  HANNOVER | VERSIE 1  VAK: DMDD  DOCENTEN: Marco Engelbart en Chris Scholten  STUDENTEN: Jasmijn Bartelds (602898), Erik Knaake (598368) en Baris Urhan (598502).  DATUM EN PLAATS: 11-04-2019 Hannover  IN OPDRACHT VAN: Hogeschool Arnhem Nijmegen (HAN), Informatica en Communicatie Academie (ICA) |

Inhoudsopgave

[Inleiding 2](#_Toc5869500)

[1. Typische eigenschappen van een document-georiënteerde database. 3](#_Toc5869501)

[2. Use cases voor document-georiënteerde database 4](#_Toc5869502)

[3. Verschillen tussen document-georiënteerde databases en relationele databases 5](#_Toc5869503)

[4. Omzetten CDM Vélib’ Métropole naar MongoDB structuur 6](#_Toc5869504)

[4.1. Data access patterns 6](#_Toc5869505)

[4.2. Toelichting op documents 7](#_Toc5869506)

[4.2.1. Subscription 7](#_Toc5869507)

[4.2.2. Subscriber 7](#_Toc5869508)

[4.2.3. Station 8](#_Toc5869509)

[4.2.4. Bike 9](#_Toc5869510)

[4.2.5. BikeRide 9](#_Toc5869511)

[4.2.6. Bill 9](#_Toc5869512)

[5. Literatuurlijst 11](#_Toc5869513)

# Inleiding

In 1970 is de relationele database ontstaan. Na jaren gebruik van dit soort databases, is er na 2000 een ander soort database ontstaan, document-georiënteerde databases (MongoDB, z.d.). Maar wat is dan zo’n document-georiënteerde database? In wat verschilt deze met een relationele database?

In dit onderzoek wordt antwoord gegeven op de bovenstaande vragen. In hoofdstuk 1 worden de typische eigenschappen van een document-georiënteerde database beschreven, hoofdstuk 2 beschrijft de use cases voor document-georiënteerde databases, hoofdstuk 3 gaat in op de verschillen tussen document-georiënteerde databases en relationele databases en hoofdstuk 4 geeft een voorbeeld hoe een conceptueel datamodel (CDM) kan worden omgezet naar een document-georiënteerde database aan de hand van voorbeeld data.

# 1. Typische eigenschappen van een document-georiënteerde database.

Dit hoofdstuk gaat in op de belangrijkste eigenschappen van een document-georiënteerde database.

Een document-georiënteerde database of document store slaat data op in verschillende documenten. Een document is vaak opgeslagen in een flexibel format zoals JSON, XML of YAML. Een van de voordelen van zo’n flexibel format is dat alleen data die beschikbaar is wordt opgeslagen, er is dus geen overhead voor het opslaan van null-waardes (*“Document-oriented database”,* z.d.). Door de eigenschap van het flexibele format, kan er ook worden gesproken van een flexibel schema.

Deze flexibele schema’s kunnen genest worden, een JSON-document kan dus een ander JSON-document bevatten, door op deze manier te nesten, of te embedden, kan de leessnelheid van een document-georiënteerde database hoog zijn omdat er in dit geval geen joins nodig zijn die een kwadratische complexiteit hebben. Hiernaast kan er door te kiezen voor embedden in plaats van een referentie voorkomen worden dat er een referentie wordt gelezen naar data die inconsistent is met de rest van het document (Matthew Revell, 2014).

Het flexibele schema zorgt er ook voor dat wanneer de eisen aan de database wijzigen het schema kan worden gewijzigd zonder dat er een complete datamigratie moet worden gedaan. Hierdoor is een document-georiënteerde database meer geschikt voor een agile team dan dat een relationele database is (Alachisoft, z.d.).

Omdat een document-georiënteerde database zich niet focust op consistentie en joins vaak op de applicatie worden gedaan en niet op de server, kan data worden verspreid over meerdere servers. Een document-georiënteerde database heeft in vergelijking met een relationele database dus een goede horizontale schaalbaarheid (IAN, 2016). Document stores zoals MongoDB doen dit automatisch doormiddel van autosharding (MongoDB, z.d.). De meeste document store hebben standaard ook ondersteuning voor replicatie, terwijl dit voor sommige relationele databases add-ons zijn (MongoDB, z.d.).

Databases kunnen nooit aan alle punten van de CAP-theorie voldoen:

* Consistency, alle data in de database komt met elkaar overeen.
* Availability, alle verzoeken tot lezen en schrijven worden in redelijke tijd verwerkt.
* Partition tolerance, het systeem blijft werken ondanks dat een deel van het systeem niet meer werkt (er valt bijvoorbeeld een node uit) of als er een message wordt gemist.

Relationele databases leggen de nadruk op consistency en availability, terwijl document-georiënteerde database, zoals MongoDB, vaak kiezen om de nadruk te leggen op consistency en partion tolerance. Een aantal document-georiënteerde databases implementeren wel het principe van evantual consistency, dit houdt in dat data contistent wordt wanneer het systeem geen nieuwe input meer ontvangt (“*ACID and CAP*”, 2016).

# 2. Use cases voor document-georiënteerde database

Een document-georiënteerde database wordt meestal gebruikt wanneer de snelheid en beschikbaarheid belangrijk zijn en het niet erg is als er incorrecte data wordt gelezen. Een document store biedt ook voordelen, wanneer er veel data is met kleine verschillen in het schema, doordat in een flexibel schema dit vaak als één document kan worden gezien. Een aantal use cases die Amazon noemt (Amazon z.d.) zijn:

* Content managementsystemen zoals blogs en social-media platformen. Hierbij is vooral de hoge beschikbaarheid en schaalbaarheid van belang.
* Catalogen van producten, waarbij verschillende soorten producten verschillende attributen hebben. Hier wordt het voordeel van het flexibele schema benut.

# 3. Verschillen tussen document-georiënteerde databases en relationele databases

Er zijn een aantal verschillen tussen document-georiënteerde databases (denk aan MongoDB, Cassandra, HBase, Neo4j) en relationele databases (denk aan MySQL, Postgres, Microsoft SQL Server, Oracle Database) (MongoDB, z.d.). Dit hoofdstuk beschrijft een aantal van deze verschillen.

Een van de voordelen van een document-georiënteerde database is dat de code die gebruikt wordt voor het schrijven van een document-georiënteerde database meer lijkt op code die in applicaties wordt gebruikt dan de code van relationele databases (Alachisoft, z.d.).

Daarnaast zijn document-georiënteerde databases gebouwd om goed te kunnen reageren op verschillende omstandigheden. Dit is terug te zien in de manier hoe de data wordt opgeslagen, namelijk in bijvoorbeeld JSON-documenten. Bij het aanmaken van een verzameling van data hoeven er geen attributen gedefinieerd worden. Hierdoor hoeft er bij een wijziging van de structuur van de data geen schema aangepast te worden, er hoeft alleen een nieuw JSON-document toegevoegd te worden. Dit heeft geen effect op de al bestaande data.

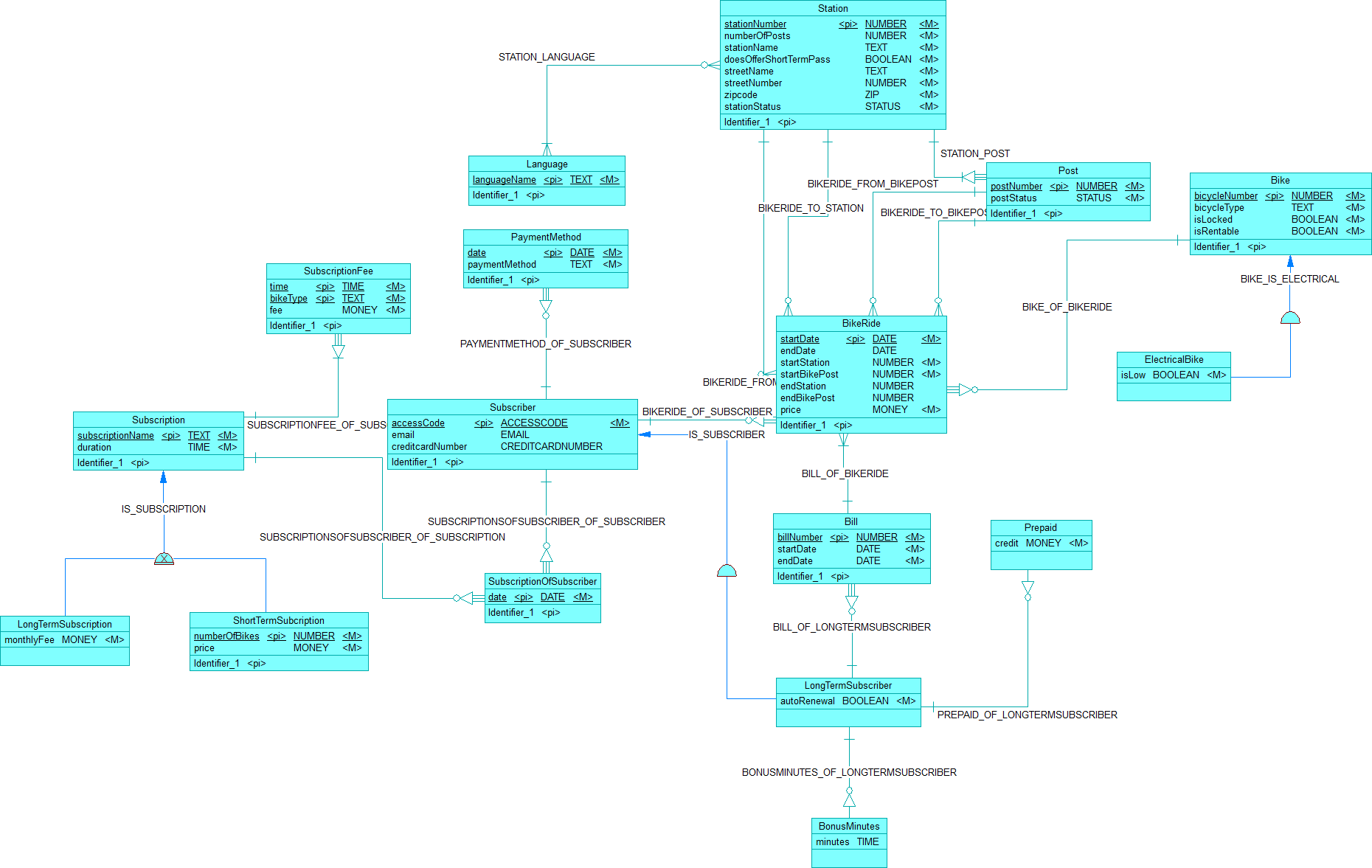
Dit is anders bij relationele databases, daarbij moeten de attributen wel gedefinieerd worden. De data wordt bij relationele databases opgeslagen in tabellen in plaats van documenten. Een wijziging in de tabel structuur zal hierbij wel effect hebben op de aanwezige data (Alachisoft, z.d.), waardoor een datamigratie nodig kan zijn.

Een ander verschil is dat bij document-georiënteerde databases de schema’s rechtstreeks gegenereerd kunnen worden vanuit de code van de applicatie, bij relationele databases kan dit niet. Bij de laatste moet dit door programmeurs zelf gedaan worden (Alachisoft, z.d.).

Ook een verschil zit in de manier hoe data uit verschillende documenten/tabellen samen opgehaald kunnen worden. Document-georiënteerde databases kunnen gebruik maken van ‘embed related documents’. Hierbij wordt verschillende data (uit verschillende documenten) die vaak samen met elkaar worden opgehaald, samengevoegd in een document. Bij relationele database kan dit niet, hierbij zal de data elke keer weer opnieuw samengevoegd moeten worden door bijvoorbeeld een JOIN. Data uit verschillende tabellen kunnen dus niet blijvend samengevoegd worden (Alachisoft, z.d.).

# 4. Omzetten CDM Vélib’ Métropole naar MongoDB structuur

In het uitwisselingsproject tussen de HAN en de HsH is er in Hannover in 2019 een CDM uitgewerkt van de casus Vélib’ Métropole – Paris by bike, deze casus gaat over het huren van fietsen in Parijs, waarbij fietsen op vele stations kunnen worden opgehaald en kunnen worden teruggebracht (HAN & HsH, 2019). Hieronder ziet u een vroege versie van het CDM van de belangrijkste onderdelen van deze casus:



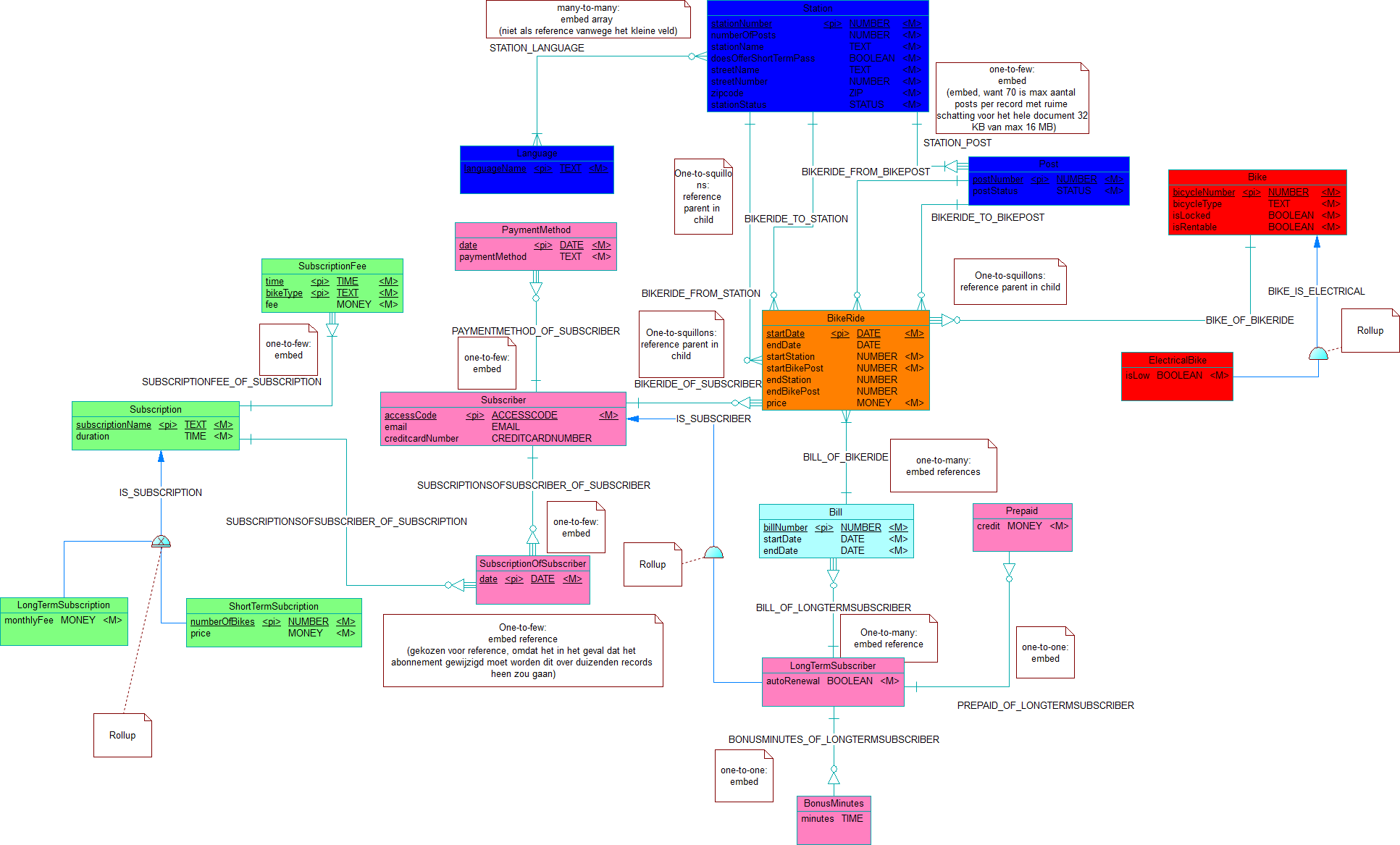
Figuur - CDM Vélib' Métropole

## 4.1. Data access patterns

Om het CDM van Vélib’ Métropole om te zetten naar een MongoDB structuur is er allereerst gekeken naar hoe data gebruikt wordt. Hieruit kwamen de volgende belangrijke punten:

* Beheren subscriptions
* Beheren subscribers
* Beheren fietsen
* Beheren stations
* Facturering

Omdat BikeRide vele relaties kent en een entity is waarna vaak wordt geschreven en er ook per insert een update moet plaatsvinden (om de endDate, endStation en endPost in te vullen) is er gekozen om voor BikeRide een eigen document te geven. Uit dit datagebruik is de volgende verdeling te maken van samenhangende entities:



Figuur - Verdeling documents

In de bovenstaande afbeelding staat elke kleur voor een document. Verder is er per relatie aangegeven of het over one-to-few, one-to-many, one-squillons of over one-to-one gaat. Afhankelijk van het soort relatie en de impact die een update heeft is er gekozen om entities te embedden of te refereren.

## 4.2. Toelichting op documents

De volgende subhoofdstukken zullen steeds ingaan op een specifiek document, waarbij belangrijke keuzes worden toegelicht en er is steeds een voorbeeld te vinden van hoe het document er uit zou kunnen zien met voorbeeld data.

### Subscription

Omdat alle entiteiten die te maken hebben met een Subscription weak zijn, kunnen al deze entiteiten worden opgenomen in één document.

{

\_id: 1,

subsciptionName: "V-MAX",

monthlyFee: 8.30,

SubcriptionFee: [

time: 30,

bikeType: "Electrical",

fee: 0.00

]

}

### Subscriber

Omdat een update in subscription voor alle subcribers alle (inclusief oude) Subscriptions van het betreffende soort zou moeten updaten en dit over vele records gaat, is er gekozen voor een minder efficiënte read om updates snel te houden en de data tussen de Subscriber en Subscription consistent te houden, door referenties op te nemen naar Subscription.

{

\_id: 100,

accessCode: 12345678,

email: "e.knaake@gmail.com",

creditcardNumber: "1234 1234 1234",

SubscriptionOfSubscriber: [

{

date: ISODate("05-03-2013 14:20:15"),

subscriptionName: 2

},

{

date: ISODate("10-04-2019 12:45:10"),

subscriptionName: 1

}

],

PaymentMethod: [

{

date: ISODate("10-04-2019 12:45:10"),

paymentMethod: "CREDITCARD"

}

],

autoRenewal: false,

bonusMinutes: {

minutes: 5

},

Prepaid: {

credit: 20.01

}

}

### Station

Er is bij het station gekozen om de Posts te embedden in hetzelfde document, omdat het onwaarschijnlijk is dat een post wordt uitgelezen zonder het station en er maximaal 70 Posts zijn per Station zijn. Uitgaande van een max van 255 bytes per record komt de totale grootte van het document rond de 32 KB, terwijl een document 16 MB groot max zijn (MongoDB, 29 mei 2014).

{

\_id: 452,

stationNumber: 5000,

numberOfPosts: 50,

stationName: "Place Valhubert",

doesOfferShortTermPass: true,

streetName: "rue pasteur"

streetNumber: 56,

zipcode: 7203AZ,

stationStatus: "open",

post: [

{

postNumber: 1,

postStatus: "blocked",

},

{

postNumber: 2,

postStatus: "free",

},

{

postNumber: 1,

postStatus: "not free",

}

],

Language: [

"Dutch",

"English"

]

}

### Bike

Omdat MongoDB een flexibel schema gebruikt is het mogelijk om het ElectricalBike subtype in het document van de Bike te zetten, zonder dat er problemen ontstaat met null-waarden.

{

bicyleNumber: 29003,

bicyleType: "Electrical",

isLocked: true,

isRentable: true,

ElectricalBike: [

{

isLow: false

}

]

}

### BikeRide

Omdat een Bike in vele BikeRides gebruikt kan worden, is er gekozen om aan de Bike te refereren vanuit de BikeRide.

{

\_id: 57424,

startDate: ISODate("05-03-2013 14:20:15"),

endDate: ISODate("05-03-2013 14:20:43"),

startStation: 452,

startBikePost: 10,

endStation 523,

endBikePost: 68,

price: 1.00,

subscriber: 100,

bike: 29003

}

### Bill

De Bill refereert aan BikeRide, omdat het onwaarschijnlijk is dat een gebruiker meer dan 4 keer per dag een BikeRide heeft en de Bill over een maand gaat, waardoor de maximum grootte van het document niet zal worden overschreven.

{

\_id: 5112,

billNumber: 5112,

startDate: ISODate("01-03-2013 00:00:00"),

endDate: ISODate("31-03-2013 23:59:59"),

BikeRides: [

57424,

58254,

60239

],

LongTermSubscriber: 100

}

# Literatuurlijst

*“ACID and CAP”*, 7 juni 2016. Geraadpleegd op 10 maart 2019 van <https://mongodbforabsolutebeginners.blogspot.com/2016/06/acid-and-cap-theroems.html>

Alachisoft, *What is a Document Database?.* z.d. Geraadpleegd op 10 maart 2019 van <http://www.alachisoft.com/nosdb/document-databases.html>

Amazon, *What Is a Document Database?*. z.d. Geraadpleegd op 10 maart 2019 van <https://aws.amazon.com/nosql/document/>

*“Document-oriented database”*. z.d. Geraadpleegd op 10 april 2019 van <https://en.wikipedia.org/wiki/Document-oriented_database>

HAN & HsH, *International Workshop ‘Information Modeling’*. 2019.

IAN, *What is a Document Store Database?* 22 juni 2016. Geraadpleegd op 10 maart 2019 van <https://database.guide/what-is-a-document-store-database/>

Matthew Revell, *Data modelling: when to embed when to refer*. 16 december 2014. Geraadpleegd op 10 maart 2019 van <https://blog.couchbase.com/data-modelling-when-embed-or-refer/>

MongoDB, *6 Rules of Thumb for MongoDB Schema Design: Part 1*. 29 mei 2014. Geraadpleegd op 14 maart 2019 van https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1

MongoDB, *NoSQL Databases Explained*. z.d. Geraadpleegd op 10 maart 2019 van <https://www.mongodb.com/nosql-explained>