

# Skalierbare Analytics: Elasticsearch im Zentrum einer BI-Infrastruktur

code.talks commerce special 2017

*Ole Golombek, CTO & Co-Founder @ minubo*

# Disclaimer

Dieser Vortrag...

- ... enthält **keinen Code**
- ... setzt ein **wenig Grundwissen** im Bereich Datenbanken bzw. Elasticsearch voraus
- ... soll interessant für Elasticsearch-**Anfänger und -Fortgeschrittene** sein
- ... zeigt, wie wir bei **minubo** Elasticsearch einsetzen (was z.B. etwas anderes sein kann als der Inhouse-Einsatz)

# minubo.com

Ein kurzer Überblick, wozu das Ganze überhaupt nötig ist

# Was ist minubo?

Eine sehr kurze Zusammenfassung



- minubo ist die **Commerce Intelligence Suite**
- Unser Produkt ist **speziell** auf die Bedürfnisse des datengetriebenen (Omni-Channel) **Handels** fokussiert
- Wir sind vor ca. 3,5 Jahren gegründet worden
- Unsere Kunden sind vor allem in Deutschland, UK und Amerika ansässig
- Unsere **Analytics-Datenbank** beinhaltet Transaktionen in einem Gesamtvolumen von mehr als 6 Milliarden €



# Fachliche Anforderungen

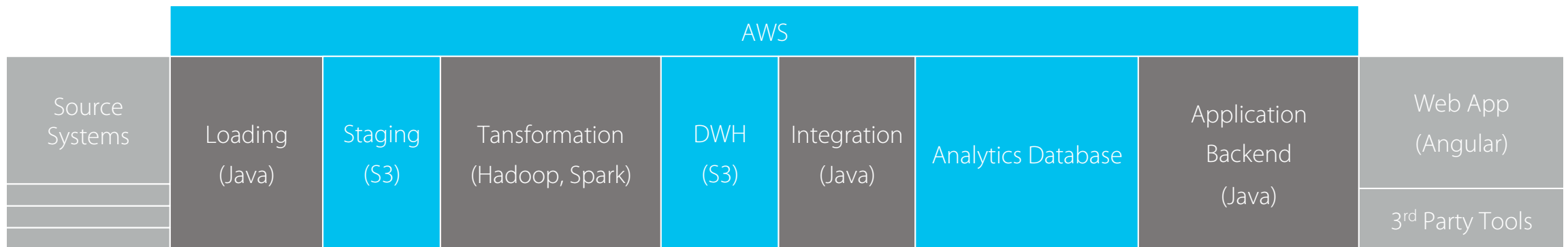
Das erwarten unsere Kunden

- **Ganzheitliche** Abbildung von Omni-Channel-Prozessen – von Transaktionsdaten über Marketingdaten bis hin zu Lagerdaten (u.v.m.)
- **Jeder Mitarbeiter** hat andere Anforderungen – von der Geschäftsführung (Übersicht) bis zum operativen Mitarbeiter (Detailsichten, Prozessunterstützung)
- **Jeder Kunde** ist anders – vom Hundefutter bis zum Luxus-Kaufhaus
- Das Ziel ist, **Wert aus Daten** zu generieren!

# Technischer Überblick


Grobe Übersicht über das minubo-Setup

- Nächtliches Update
- ~2 Milliarden Datensätze in der Datenbank
- Web App und externe Tools haben volle **Flexibilität** bezüglich Datenabfragen (~200 Attribute, ~400 Kennzahlen)
- Abfrageergebnisse müssen in **Real-Time** zur Verfügung stehen



# app.minubo.com

## Die Commerce Intelligence Suite



Heartbeat <sup>13</sup>

Dashboards

Analyses

Segmentations

Reporting

Feeds

More

What are you looking for?

Help

Account

Heartbeat > Current

Today

Reactivation Opportunities (1)

24 customers show irregular buying behavior and should be targeted now

✓ ?

High return rate (1)

1 products have unusual high return rates.

✉ ✓ ?

Data Import

Your data has been imported successfully by 01:15 PM

Yesterday

Channel Performance (2)

The following channels deviate from the usual weekday average: EBAY, WEBSHOP

✓ ?

Performance of channel roadborne.com | EBAY has changed significantly.

✉ ✓

Gross Order Value after Discount	Gross Order Number	Avg Order Value (incl VAT, ex Discounts)
+220 %	+60.40 %	+39.83 %
€198.91	3	€66.30

Show Drivers

Performance of channel roadborne.com | WEBSHOP has changed significantly.

✉ ✓

Gross Order Value after Discount	Visits	Conversion Rate (Web)	Avg Order Value (incl VAT, ex Discounts)
+226 %	N/A	N/A	+32.31 %

# Elasticsearch

Warum Elasticsearch?



# Elasticsearch

You Know, for Search...

- Open-Source Search Engine
- Distributed by design (und damit skalierbar)
- NoSQL
- RESTful
- Keine Transaktionen
- Basiert auf Lucene (Java-Bibliothek zur Volltextsuche)

# Elasticsearch

... and for Analytics!

- Von der Ermittlung von Trefferzahlen pro Keyword ist es nicht mehr weit zur **Aggregation**
- In den letzten Jahren hat sich Elasticsearch zur **Analytics-Lösung** entwickelt

# Erste Performance-Tests

Elasticsearch ist schnell

- Hervorragende erste Performance-Kennzahlen (Testdatenset: ~80 M Visits)
- Sehr große Datenmengen können sehr schnell aggregiert werden
- Vergleich mit parallel getesteter spaltenbasierter Datenbank\* eindeutig
- Gerade im Filtern sehr stark

\*) getestet wurden neben Infobright weitere Datenbanken, die in unserem Use Case aber schlechter abschnitten

Request	Result	
	Database	Time
<pre>SELECT Date, COUNT(*) AS VisitCount FROM Visits GROUP BY Date ORDER BY VisitCount DESC LIMIT 10</pre> <pre>SELECT Date, COUNT(*) AS VisitCount FROM Visits WHERE City = 'Hamburg' GROUP BY Date ORDER BY VisitCount DESC LIMIT 10</pre>	Infobright	~8 Sec
	Elasticsearch	~1 Sec
	Infobright	~33 Sec
	Elasticsearch	~0,08 Sec

# Woher kommt die Performance?

## Elasticsearch-Grundlagen – verschiedene Speicherformen

### - Index (Fielddata)

- name: 0 = Ole / 1 = Ole / 2 = Anne / 3 = Roman / 4 = Matthias
- department: 0 = Development / 1 = Development / 2 = Sales / 3 = Development / 4 = Support

### - Inverted Index

- name: Anne = 2 / Matthias = 4 / Ole = 0,1 / Roman = 3
- department: Development = 0,1,3 / Sales = 2 / Support = 4

### - Documents

- 0: name = Ole / department = Development
- 2: name = Anne / department = Sales
- (nutzen wir nicht)

Index (Documents)

```
0: {  
  "date": "2017-04-20",  
  "name": "Ole",  
  "department" : "Development",  
  "coffees": 4  
}  
1: {  
  "date": "2017-04-21",  
  "name": "Ole",  
  "department" : "Development",  
  "coffees": 5  
}  
2: {  
  "date": "2017-04-20",  
  "name": "Anne",  
  "department" : "Sales",  
  "coffees": 3  
}  
3: {  
  "date": "2017-04-20",  
  "name": "Roman",  
  "department" : "Development",  
  "coffees": 8  
}  
4: {  
  "date": "2017-04-21",  
  "name": "Matthias",  
  "department" : "Support",  
  "coffees": 1  
}
```

# Woher kommt die Performance?

## Elasticsearch Grundlagen – Sharding

- Documents werden auf verschiedene **Shards** aufgeteilt
- Shards werden bei einer Anfrage zunächst **unabhängig** voneinander ausgewertet
- Das Gesamtergebnis wird am Ende ermittelt

Shard A	
Bestellung	Bestellwert
Order A.1	30
Order B.1	20
Order B.2	45
95	

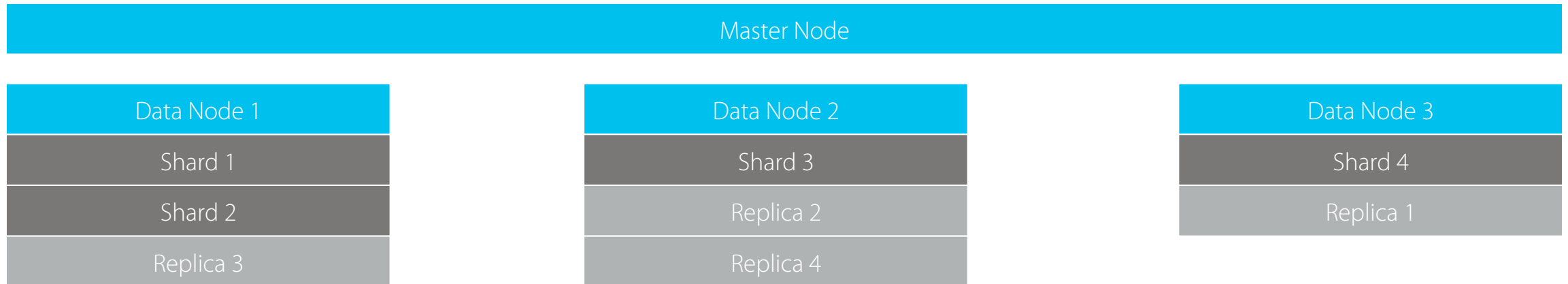
Master	
Shard A	95
Shard B	125
230	

Shard B	
Bestellung	Bestellwert
Order A.2	15
Order A.3	100
Order C.1	10
125	

# Sharding führt zu Skalierbarkeit

Shards können auf verschiedene Nodes verteilt werden

- Elasticsearch kümmert sich um das Routing
- Shard **Replicas** erhöhen Ausfallsicherheit
- Aber: Elasticsearch-Cluster sind „**lebendig**“



# Einfache Summierung

Ein kurzer Einblick, wie in Elasticsearch Aggregationen funktionieren

Request

```
POST /DEMOSHOP_events/_search
{
  "size" : 0, //aggregation only, no document results
  "aggregations" : {
    "grsOrdVal" : { //gross order value
      "sum" : {
        "field" : "grsOrdVal"
      }
    }
  }
}
```

Result

```
{
  "took": 181, ... //milliseconds
  "hits": {
    "total": 52943746, ... //documents in index
  },
  "aggregations": {
    "grsOrdVal": {
      "value": 88076031.66211024
    }
  }
}
```

Request

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "genders" : {
      "terms" : { //group by
        "field" : "cstGender" //customer gender
      },
      "aggs" : {
        "grsOrdQty" : { //gross order quantity
          "sum" : {
            "field" : "grsOrdQty"
          }
        }
      }
    }
  }
}
```

Result

```
{
  "took": 170, ...
  "aggregations": {
    "genders": {
      "buckets": [
        {
          "key": "M",
          "doc_count": 817709,
          "grsOrdQty": {
            "value": 91143
          }
        },
        {
          "key": "F",
          "doc_count": 808441,
          "grsOrdQty": {
            "value": 90051
          }
        }
      ]
    }
  }
}
```

# Distinct Count

## Ein Problemfall

- Distinct Counts sind in vielen Kennzahlen unverzichtbar
- Leider sind sie ein Problem für den Arbeitsspeicher – auch in verteilten Systemen
- Der Master muss eine Gesamtliste aller Werte führen

Shard A		
Bestellung	Kunde	
Order A.1	Customer A	A
Order B.1	Customer B	B
Order B.2	Customer B	B

Master		
Shard A	A	A
Shard A	B	B
Shard B	A	A
Shard B	C	C
		3

Shard B		
	Bestellung	Kunde
A	Order A.2	Customer A
A	Order A.3	Customer A
C	Order C.1	Customer C



# Cardinality

Wie Elasticsearch Distinct Counts anbietet

- Elasticsearch bietet **Cardinality** Aggregation
- Nutzt **HyperLogLog++** Algorithmus
- Dieser schützt den Arbeitsspeicher auf Kosten von Genauigkeit
- Das akzeptieren die Nutzer leider nicht

Request

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "grsOrdNum" : { //Anzahl Bruttobestellungen
      "cardinality" : {
        "field" : "grsOrdId" //Hashkey pro Bruttobestellung
      }
    },
    "grsOrdNumPrecison" : {
      "cardinality" : {
        "field" : "grsOrdId",
        "precision_threshold": 5000
      }
    }
  }
}
```

Result

```
{
  "took": 382, ...
  "aggregations": {
    "grsOrdNum": {
      "value": 82605
    },
    "grsOrdNumPrecison": {
      "value": 83021
    }
  }
}
```

# Elasticsearch-Plugins

Was nicht passt, wird passend gemacht

# Real Distinct Count

Plugin, das echten Distinct Count unterstützt

- In Java geschriebenes Aggregations-**Plugin**
- Eigener Algorithmus, der Array-Kopier-Aufwände minimiert (angelehnt an HashSet, optimiert auf Longs)
- Performance ist nahe an Cardinality
- Gefahr: Speicherüberlauf! (Aber wir kennen unsere Daten)

Request

```
GET /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "grsOrdNumDC" : {
      "distinctCount" : {
        "field" : "grsOrdId"
      }
    }
  }
}
```

Result

```
{
  "took": 311, ... //vs 220 cardinality 40000 (max)
  "aggregations": {
    "grsOrdNumDC": {
      "value": 82977 //vs 82851
    }
  }
}
```

# Shard Distinct Count

Verbesserung durch „Smart Sharding“

- Moment: Wir kennen unsere Daten wirklich
- Sharding Rule: Ein **Kunde** ist immer auf einem **Shard**
- Gefahr: Sehr große Kunden stören das Shard-Gleichgewicht
- Performance-Gewinn: **keine Sync** zwischen den Shards mehr **nötig**
- Außerdem: weniger Arbeitsspeicher nötig

Shard A		
Bestellung	Kunde	
Order A.1	Customer A	A
Order A.2	Customer A	A
Order A.3	Customer A	A
		1

Master	
Shard A	1
Shard B	2
	3

Shard B		
	Bestellung	Kunde
B	Order B.1	Customer B
B	Order B.2	Customer B
C	Order C.1	Customer C
2		

# Distinct Count Vergleich

Die Unterschiede im Überblick

## Request

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "grsOrdNum" : {
      "cardinality" : {
        "field" : "grsOrdId", "precision_threshold": 40000
      }
    }
  }
}
```

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "grsOrdNumDC" : {
      "distinctCount" : {
        "field" : "grsOrdId"
      }
    }
  }
}
```

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "grsOrdNumDC" : {
      "shardDistinctCount" : {
        "field" : "grsOrdId"
      }
    }
  }
}
```

## Result

```
{
  "took": 220, ...
  "aggregations": {
    "grsOrdNumDC": {
      "value": 82851
    }
  }
}
```

```
{
  "took": 311, ...
  "aggregations": {
    "grsOrdNumDC": {
      "value": 82977
    }
  }
}
```

```
{
  "took": 258, ...
  "aggregations": {
    "grsOrdNumDC": {
      "value": 82977
    }
  }
}
```

# Berechnete Kennzahlen

In der Kombination mit Sortierung ein Problem

# Berechnete Kennzahlen

Betrifft vor allem Quoten

- Bruttobestellwert pro Bestellung =  $\text{Bruttobestellwert} / \text{Bestellungen}$
- Formel:  $\text{avgGrsOrdVal} = \text{SUM}(\text{grsOrdVal}) / \text{DC}(\text{grsOrdId})$
- Möglichkeit A: Einzelwerte von Elasticsearch berechnen lassen, Berechnung anschließend in nutzender Schicht durchführen
- Problem: Sortierung
- Möglichkeit B: Plugin-Erweiterung

# Berechnete Kennzahlen

## Formel-Interpreter als Aggregations-Plugin

- Unterstützt Grundrechenarten auf SUMs, DCs und SDCs (Shard Distinct Count)
- Performance nicht schlechter als Einzel-Aggregationen

Request

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "avgGrsOrdVal" : {
      "formula" : {
        "formula" : "SUM[grsOrdVal] / SDC[grsOrdId]"
      }
    }
  }
}
```

Result

```
{
  "took": 428, ... //vs. 405 sum + shardDistinctCount
  "aggregations": {
    "avgGrsOrdVal": {
      "value": 1061.4511450415205
    }
  }
}
```



# Sortierung auf berechnete Kennzahlen

Funktioniert!

- Achtung – grundsätzlich besteht hier noch ein weiteres Problem: **Sortierung** braucht **viel Arbeitsspeicher** (bzw. Elasticsearch wird irgendwann ungenau)
- Lösung zunächst: Sortierung nur auf kleinen Sets zulassen

## Request

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "countries" : {
      "terms" : {
        "field" : "cstCountry",
        "order" : { "avgGrsOrdVal" : "desc" }
      },
      "aggs" : {
        "avgGrsOrdVal" : {
          "formula" : {
            "formula" : "SUM[grsOrdVal] / DCP[grsOrdId]"
          }
        }
      }
    }
  }
}
```

## Result

```
{
  "took": 148, ... //only docs with country considered
  "aggregations": {
    "countries": {
      "buckets": [
        {
          "key": "Australia",
          "avgGrsOrdVal": {
            "value": 1348.7646002381557,
          }
        },
        {
          "key": "Germany",
          "avgGrsOrdVal": {
            "value": 1165.1821007246297,
          }
        },
        ...
      ]
    }
  }
}
```

# Mehrfach genutzte Basiswerte

## Optimierung Multiformula

- Um **Basiswerte** nicht **mehrfach** berechnen zu müssen, kommt das Multiformula-Plugin hinzu
- Wertet im Hintergrund alle Formeln aus und berechnet Werte nur einmal

### Formula

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "avgGrsOrdVal" : {
      "formula" : {
        "formula" : "SUM[grsOrdVal] / SDC[grsOrdId]"
      }
    },
    "avgGrsOrdQty" : { //Bruttobestellmenge pro Bestellung
      "formula" : {
        "formula" : "SUM[grsOrdQty] / SDC[grsOrdId]"
      }
    }
  }
}
```

```
{
  "took": 914
}
```

### Multiformula

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "values" : {
      "multiformula" : {
        "avgGrsOrdVal" : "SUM[grsOrdVal] / SDC[grsOrdId]",
        "avgGrsOrdQty" : "SUM[grsOrdQty] / SDC[grsOrdId]"
      }
    }
  }
}
```

```
{
  "took": 538
}
```

# Einbettung ins System

Nutzung der Datenbank

# Aggregation API

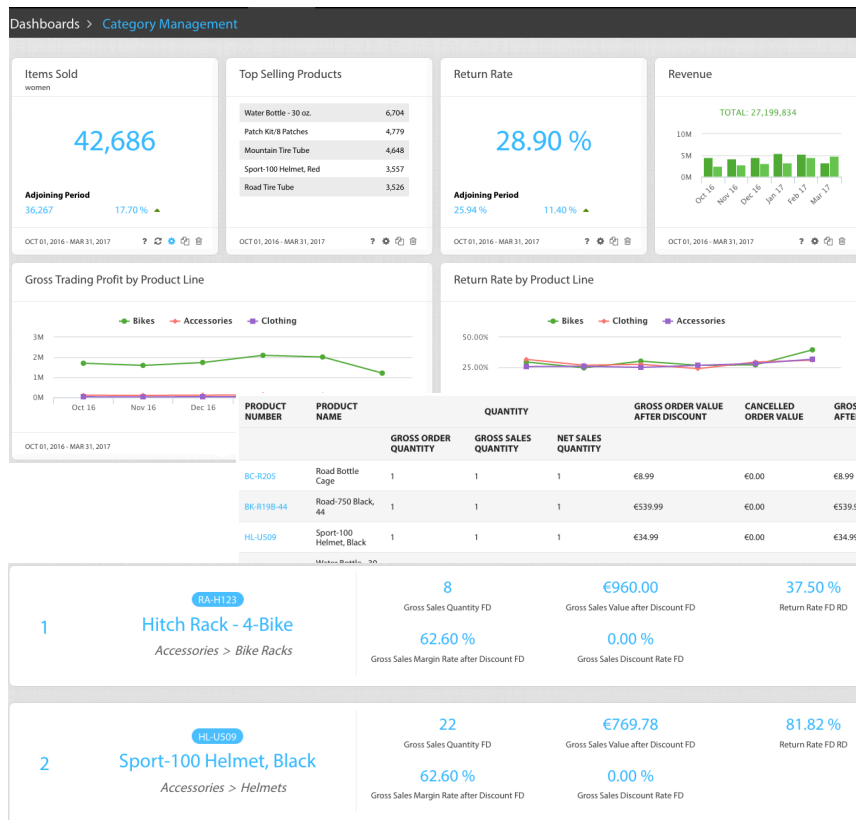
Abstraktionsebene und Modell-Definition

- Abfragen aus Java Backend über **Abstraktionsebene** „Aggregation API“
- **Kapselt** Elasticsearch und macht dieses damit austauschbar
- Wesentliche **Vereinfachung**, da Suchoptionen ausgeblendet werden
- Kennzahlen- und Attributsdefinitionen sind im Code hinterlegt
- Auch **Metadaten** (z.B. Verknüpfbarkeit und Berechnungsgrundlagen) sind ins Modell integriert, womit zum einen Hilfetexte generiert und zum anderen automatische Tests durchgeführt werden können

# Die App

Darstellung der Daten in Dashboards, Reports, Web-Pivot und für Alerts

- Die App nutzt die Datenbank für **verschiedenste Analysemöglichkeiten**



		DAY						
GROUP	METRIC	26.03.2017	27.03.2017	28.03.2017	29.03.2017	30.03.2017	31.03.2017	TOTAL
Traffic	Marketing Impressions	112,760	112,760	112,760	112,760	112,760	112,760	676,560
	Marketing Click-Through-Rate	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
	Clicks	0	0	0	0	0	0	0
	Visits	0	0	0	0	0	0	0
	Number of Unique Visitors	0	0	0	0	0	0	0
	Conversion Rate (Web)	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
	Gross Order Number	129	73	53	64	60	56	435
Orders	Average Orders per Customer	1.02	1.00	1.00	1.00	1.00	1.00	1.02
	Number of Customers	127	73	53	64	60	56	426
	Number of New Customers	36	0	0	0	0	0	36
	Average Order Quantity	GROSS ORDER VALUE						
	Gross Order Quantity	2017-04-18						
	Average Order Value	2017-04-19						
	Gross Order Discount	2017-04-20						
Revenue	Gross Order Value after	2017-04-21						
	Cancellation Rate (Val	2017-04-22						
	Fulfillment Rate	2017-04-23						
	Gross Sales Value after							
		AWC LOGO CAP						
		ALL-PURPOSE BIKE STAND						
		BIKE WASH - DISSOLVER						
		CHAIN						
		CLASSIC VEST, L						
		CLASSIC VEST, M						
		CLASSIC VEST, S						
		FENDER SET - MOUNTAIN						

# Feeds

## Ausgehende Schnittstelle

- Macht **Export** beliebiger Abfragen mit **mehreren Millionen Zeilen** möglich
- Dabei hilft uns, dass das Ergebnis **unsortiert** sein kann
- Trick: Ergebnis wird in **Blöcke** unterteilt

1

`N = Schätzwert für Zeilenanzahl`

2

`B = N / 10000 = Anzahl Blöcke (10000 als Blockgröße)`

3

`b = 0..(B - 1) = Blockindex`

4

`d = Dokument`

5

`A = Attribute der Abfrage`

6

`k(d, A) = Konkatination aller Werte der Abfrage-Attribute eines Dokuments`

7

`h(k) = Hashwert von k`

8

`db = mod(h, B) = Block des Dokuments`

9

`x = db == b = Entscheidungskriterium, ob Dokument d in der Abfrage b berücksichtigt wird`

# Kundensegmentierung

Bedingte Aggregationen machen flexible Segmentierung möglich

- **Bedingte Formeln**, Filter auf Buckets, kundenbasiertes Sharding sowie Blockbildung erlauben **Adhoc-Berechnung von Segmenten** auf Basis von Kauf- und Marketinghistorie

Request (Vereinfacht)

```
POST /DEMOSHOP_events/_search
{
  "size" : 0,
  "aggregations" : {
    "customers" : {
      "terms" : {
        "field" : "cstKey",
        "filter" : { "bigger" : { "field": "slsVal2016Bikes", "value": 2000 } }
      },
      "aggs" : {
        "slsVal2016Bikes" : {
          "formula" : {
            "formula" : "SUM[slsVal]",
            "filter" : {
              "dateRange": { "from" : "2016-01-01", "to": "2016-12-31" },
              "terms": { "field" : "prdCat", "values" : ["Bikes"] }
            }
          }
        }
      }
    }
  }
}
```

## Rules

Filter by Customer Attributes

**Gender** equals **M**.

Filter by Transaction History

**Time:** Between **Apr 25, 2016** and **Apr 24, 2017** (Last 365 Days).

**Product Category** equals **Mountain Bikes**.

**Gross Sales Value** is less than **€1,000**.

Filter by Transaction History

**Time:** Between **Mar 26, 2017** and **Apr 24, 2017** (Last 30 Days).

**Gross Order Quantity** equals **0**.

# Next

Wie geht es weiter?



# Weitere Entwicklung

Wir entwickeln unsere Analytics Cloud ständig weiter

- **Real-Time**: Zusätzlicher Index mit ausgewählten Kennzahlen und Attributen, die ständig geupdatet werden
- **Delta-Update**: Bisher wird die Datenbank jede Nacht komplett neu erstellt – wir wollen auf ein Delta-Verfahren umstellen
- **Flexiblere Skalierung**: Ressourcensparend könnten Nodes bei wenig Last heruntergefahren werden
- Und, und, und ...

~~Don't try to~~ *You don't have to*  
do this at home!

*minubo.com*

*Ole Golombek, CTO & Co-Founder*

*ole@minubo.com*