|  |  |
| --- | --- |
| Bike Shop | Meilenstein 3  Thomas Rauchbauer (11740131) Ilhan Angin (01576537) |

# Motivation

Wir haben uns für die Umstellung auf NOSQL entschieden, da die Datenmengen immer größer werden. Unser Bike Shop möchte den Online Shop vergrößern und die Kaufentscheidungen der Kunden speichern und Analysen. Dies spiegelt sich auch in den Designentscheidungen wieder. Die Einkäufe der Kunden sollen leicht statistisch auswertbar sein. Zusätzlich können wir uns mit neuen Standorten leicht vernetzen. Im Großen und Ganzen will das Unternehmen auf die Menge von Daten vorbereitet sein und mit die Umstellung früh genug wagen.

# Designentscheidungen

* Shop – Bike Service : One-to-One
* Bike Service – Bike: One-to-“actually many”
* Customer – Bike: One-to-“a few”
* Customer – Accessories: One-to-“a few”
* Shop – Employee: One- to- “actually many”
* Employee – Employee: One-to-“actually many”
* Employee – Accessories: One- to –“really a lot”
* Employee – Bike: Many-to-Many

## Shop – Bike Service (One-to-One)

Der Bike Service ist eine schwache Entität, deshalb haben wir uns entschieden diese Datensätze zusammenzufügen. Die daraus entstehende Collection hat den Name „Shop“. Denn ohne Shop gibt es keinen Service.

## Bike Service – Bike (One-to- “actually many”)

Wir haben uns dafür entschieden ein Array von BikeIDs im Shop zu speichern. Denn der Platz in einem Shop ist begrenzt und deshalb sollte es nicht zu mehreren Arrays führen.

### Alternative

Die Bike Services sind stark verbunden mit dem Shop deshalb könnten wir die Bikes in den Shops speichern. Jedoch würde das zu viel Speicherplatz verbrauchen.

## Die Vererbung von Person zu Customer und Employee:

Für diesen Fall haben wir uns entschieden, dass wir die Attribute von Person zu Employee und Customer hinzufügen.

CUSTOMER – BIKE ONE-TO-“A FEW”

Da wir die Bikes noch an mehreren Stellen benötigen vergeben wir auch hier wieder eine Referenz um die Informationen nicht doppelt zu speichern.

ALTERNATIVE

Eine Alternative wäre die Fahrräder zusammen mit dem Kunden zu speichern. Zusätzlich findet man alle Fahrräder des Kunden und deren Historie in einem Dokument. Jedoch Sehr viel mehr Speicheraufwand.

## Customer – Accessoires One-to-“a few”

Die Einkäufe eines Kunden in unserem Fahrradshop betragen im Durchschnitt nicht über 10 Produkte, daher speichern wir die Artikel direkt bei dem Kunden. Diese Artikel ändern sich auch nicht nach dem Einkauf daher kommt es auch zu keinen Updateaufwand.

### Alternative

Alternative hätten wir eine extra Collection für Accessories erstellen können und anhand der ID eine Referenzierung im Customer erstellen

## Shop – Employee One- to- “actually many”

In User Case 5 wird beschrieben wie der Arbeitsort eines Employee herausgefunden wird. Daher haben wir darauf Wert gelegt den zugehörigen Shop einfach aufzufinden. Wir haben uns dafür entschieden im Employee die ShopID des zugehörigen Arbeitgebers zu speichern.

### Alternative

Wir haben uns gegen Two-Way Referencing entschieden da die Updatekosten für uns zu hoch wären.

## Employee – Employee One-to-“actually many”

Jeder Employee wird nur einen Chef haben, deshalb speichern wir diesen Chef gleich komplett im Employee um die Informationsabfrage über den Chef zu verschnellern .

### Alternative

Eine andere Methode die sinnvoll gewesen wäre nur eine Referenz des Chefs zu speichern. Nachteil daran wäre mehr Aufwand bei Abfragen.

## Employee – Accessories One- to –“really a lot”

Da die Accessories im Kunden gespeichert werden wollten wir eine doppelte Speicherung vermeiden und haben uns entschieden gleich in diesen Accesories eine EmplyeeID hinzuzufügen. Anhand dieser ID kann erkannt werden vom welchem Mitarbeiter dieser Artikel versendet worden ist. Dies bringt uns außerdem einen großen Vorteil bei dem Use Case „Report“.

## Employee – Bike Many-to-Many

Hier haben wir uns für eine zweiseitige Referenzierung entschieden, da es damit zu einer Übersichtlichen Darstellung von der many to mayn Beziehung kommt.

### Alternative

Ein Alternative wäre gewesen die ganzen Entitäten zu speichern, jedoch wäre das mehr Speicheraufwand und Redundanz.

Sharding of NoSQL DBMS

Wie in der Motivation erwähnt wollen wir das Unternehmen expandieren und daher ist uns das Thema Sharding sehr wichtig.

**Sharding Strategie:** Ranged Sharding

* keine High Shard Key Frequency
* Monotonically Changing Values werden nicht benötigt, weil die Daten mit benachbarten Keys nicht voneinander Abhängen
* Mehre Chunks werden benötigt aufgrund der vielen vorgesehenen Standorten