

System Architektur v0.0.1

24.09.16 - Andrei

Überblick

1. mongodb konfigurieren
 - Sensordaten simulieren
2. statische Datenbank Schema
 - OSM Daten abholen
3. Parken Abfragen

mongodb

- Darein werden Sensordaten abgespeichert:
 - Sensor id
 - Timestamp
 - Parkplatz
 - isFree – ist jetzt der Platz frei oder belegt? (TRUE oder FALSE)

Sensordaten simulieren

- Je nach Speicherart unterscheiden sich folgende Strategien:
 - Pro Event (aktuelle Wahl)
 - Pro Sekunde
 - Falls mehrfache Events pro Sekunde
 - Vorteil: weniger Inserts, spart Schreiben-Zeit
 - Zehntelsekunde-Events sind in Gruppen gespeichert: 0..9
 - Pro Minute
 - Sekunde-Events 0..59 sind zusammen gespeichert

Mongodb - Screenshot

The screenshot displays the Robomongo 0.9.0-RC10 application window. The interface is divided into several sections:

- Left Sidebar:** A tree view showing the database structure. It includes 'System', 'mynewdb', 'sonah-live', 'Collections (1)' (containing 'sensordata'), 'Functions', 'Users', and 'test'.
- Command Editor:** A text area at the top right containing the command `db.getCollection('sensordata').find({})`. Below it, a status bar indicates 'sensordata' and '0.006 sec.'.
- Results Table:** A table with three columns: 'Key', 'Value', and 'Type'. It lists 27 documents, each with an ObjectId key and a 5-field value. The 10th document is expanded, showing its fields: `_id` (ObjectId), `sensor_id` (Int32: 137), `timestamp` (Int64: 1474713218868), `parkingplace` (Int32: 54), and `isFree` (Boolean: false).
- Bottom Bar:** A 'Logs' tab is visible at the bottom left.

Key	Value	Type
(1) ObjectId("57e501b462ba2e37c4377807")	{ 5 fields }	Object
(2) ObjectId("57e501b562ba2e37c4377808")	{ 5 fields }	Object
(3) ObjectId("57e6567b62ba2e0bf0af1276")	{ 5 fields }	Object
(4) ObjectId("57e6567c62ba2e0bf0af1277")	{ 5 fields }	Object
(5) ObjectId("57e6567d62ba2e0bf0af1278")	{ 5 fields }	Object
(6) ObjectId("57e6567e62ba2e0bf0af1279")	{ 5 fields }	Object
(7) ObjectId("57e6567f62ba2e0bf0af127a")	{ 5 fields }	Object
(8) ObjectId("57e6568062ba2e0bf0af127b")	{ 5 fields }	Object
(9) ObjectId("57e6568162ba2e0bf0af127c")	{ 5 fields }	Object
(10) ObjectId("57e6568262ba2e0bf0af127d")	{ 5 fields }	Object
_id	ObjectId("57e6568262ba2e0bf0af127d")	ObjectId
sensor_id	137	Int32
timestamp	1474713218868	Int64
parkingplace	54	Int32
isFree	false	Boolean
(11) ObjectId("57e6568362ba2e0bf0af127e")	{ 5 fields }	Object
(12) ObjectId("57e6568462ba2e0bf0af127f")	{ 5 fields }	Object
(13) ObjectId("57e6568562ba2e0bf0af1280")	{ 5 fields }	Object
(14) ObjectId("57e6568662ba2e0bf0af1281")	{ 5 fields }	Object
(15) ObjectId("57e6568762ba2e0bf0af1282")	{ 5 fields }	Object
(16) ObjectId("57e6568862ba2e0bf0af1283")	{ 5 fields }	Object
(17) ObjectId("57e6568962ba2e0bf0af1284")	{ 5 fields }	Object
(18) ObjectId("57e6568a62ba2e0bf0af1285")	{ 5 fields }	Object
(19) ObjectId("57e6568b62ba2e0bf0af1286")	{ 5 fields }	Object
(20) ObjectId("57e6568c62ba2e0bf0af1287")	{ 5 fields }	Object
(21) ObjectId("57e6568d62ba2e0bf0af1288")	{ 5 fields }	Object
(22) ObjectId("57e6568e62ba2e0bf0af1289")	{ 5 fields }	Object
(23) ObjectId("57e6569062ba2e0bf0af128a")	{ 5 fields }	Object
(24) ObjectId("57e6569162ba2e0bf0af128b")	{ 5 fields }	Object
(25) ObjectId("57e6569262ba2e0bf0af128c")	{ 5 fields }	Object
(26) ObjectId("57e6569362ba2e0bf0af128d")	{ 5 fields }	Object
(27) ObjectId("57e6569462ba2e0bf0af128e")	{ 5 fields }	Object

Statische Datenbank – Schema (1)

```
CREATE TABLE `sensors` (  
    `SensorID` bigint(20) NOT NULL AUTO_INCREMENT,  
    `Latitude` double NOT NULL,  
    `Longitude` double NOT NULL,  
    `Altitude` double NOT NULL DEFAULT '0',  
    `ParkingAreaID` bigint(20) COMMENT 'Parking area to which it belongs',  
    `Added` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
    `Type` varchar(30) COMMENT 'Type of sensor (optical, etc.)',  
    PRIMARY KEY (`SensorID`)  
);
```

Statische Datenbank – Schema (2)

```
CREATE TABLE `parkingareas` (  
    `ParkingAreaID` bigint(20) NOT NULL AUTO_INCREMENT,  
    `ParkingAreaName` varchar(30),  
    `Access` varchar(20) COMMENT 'E.g. public, customers, residents, employees',  
    `Type` varchar(20) COMMENT 'E.g. surface, multi-storey, underground, rooftop, sheds,  
garage_boxes',  
    `FeeID` bigint(20) NOT NULL,  
    /* time interval when open - missing, mysql has no such inbuilt type*/  
    `MaxHeight` double,  
    PRIMARY KEY (`ParkingAreaID`)  
);
```

Statische Datenbank – Schema (3)

```
CREATE TABLE `parkingspace` (  
    `ParkingSpaceID` bigint(20) NOT NULL AUTO_INCREMENT,  
    `Latitude` double NOT NULL,  
    `Longitude` double NOT NULL,  
    `Altitude` double NOT NULL DEFAULT '0',  
    `ParkingAreaID` bigint(20) NOT NULL COMMENT 'Parking area to which it belongs',  
    `SensorID` bigint(20) COMMENT 'Sensor to which it belongs, can be NULL',  
    PRIMARY KEY (`ParkingSpaceID`)  
);
```


Statische Datenbank – Schema (4)

```
CREATE TABLE `Fees` (  
    `FeeID` int(11) NOT NULL AUTO_INCREMENT,  
    /* time interval -- missing, mysql has no such inbuilt type */  
    `CostPerHour` double,  
    /* other type of cost?! */  
    PRIMARY KEY (`FeeID`)  
);
```

OSM Daten

- Daten werden durch Abfragen an OverpassAPI geholt
 - Parkautomaten (vending = parking_tickets)
 - Parkräume (amentiy = parking)
- Aktuell
 - `sensors` Tabelle wird mit Parkautomaten-Daten befüllt
 - `parkingareas` Tabelle wird mit Parkräume-Daten befüllt
 - `sensor (n) : parkingarea (1)` -> in OSM schwierig festzulegen!!

Parken Abfragen

- Einfache Variante: Finde die besten Parkmöglichkeiten in der Nähe von Position (lat, lon), jetzt!
- Vorgeschlagener Algorithmus:
 - Suche im Umkreis 500m
 - finde die Parkräume, die den aktuellen Standort geometrisch enthalten
 - Berechne Anteil an freien Plätzen für diese Parkräume (stand: aktuell)
 - Erstelle eine Rangliste; E.g. P1: 35% frei, P2: 24% frei, P3: 10% frei
 - Falls nicht genug Parkräume gefunden/die Raten zu niedrig sind
 - Suche im Umkreis 1000m
 - das gleiche
 - ...