ISEngineering

Technische Universität Berlin

CSEM: Database Programming

Dr. Markus Klems, Prof. Stefan Tai

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Learning objectives

Programming with …

- Relational databases
  - Data modeling
  - Database programming
  - Programming PostgreSQL-backed applications

- NoSQL databases
  - Short overview
  - Document-oriented data modeling and programming
  - Programming MongoDB-backed applications

PROGRAMMING WITH RELATIONAL DATABASES

Wirtschaftsinformatik –
Information Systems Engineering

Database Programming Techniques

Approaches to Database Programming:

1. Embedding database commands (SQL) in a general-purpose programming language.
2. **Using a library** of database functions or classes.
3. Designing a database programming language from scratch.

R. Elmasri and S. Navathe: Fundamentals of Database Systems, 7th edition, Pearson. Chapter 10.

Impedance Mismatch

Impedance mismatch is the term used to refer to the problems that occur because of differences between the database model and the programming language model.

For example, the practical relational model has three main constructs:

- columns (attributes) and their data types
- rows (tuples, records), and
- tables (sets or multisets of records).

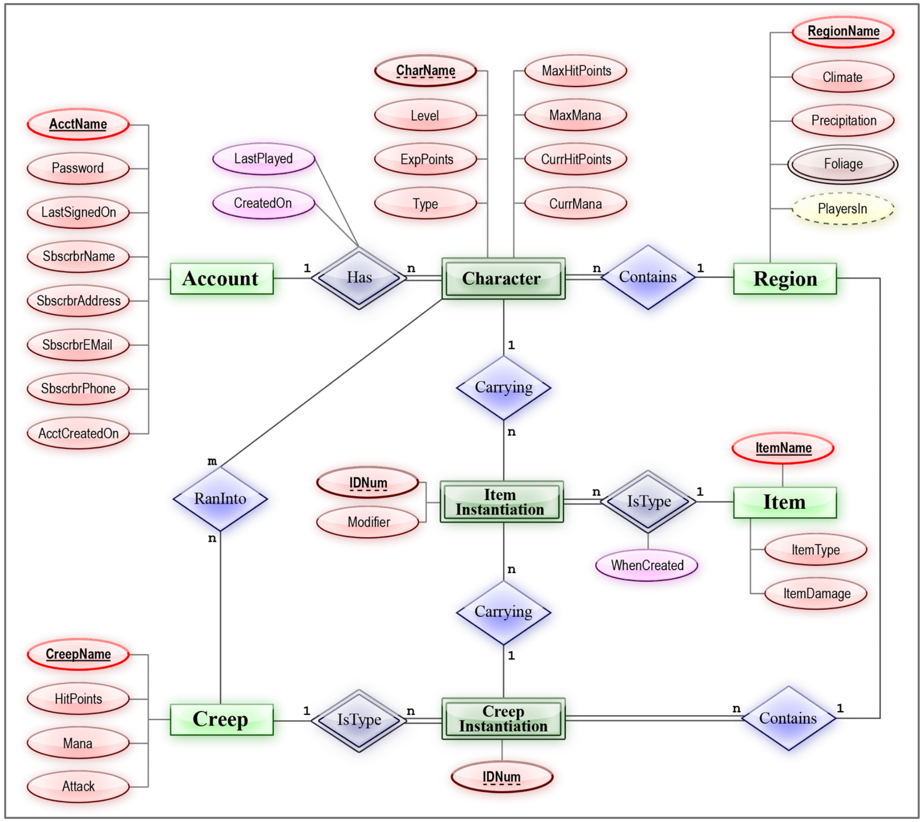
Application-level programming languages offer a different (usually object-oriented) programming paradigm with different data modeling techniques, data types and query interfaces.

R. Elmasri and S. Navathe: Fundamentals of Database Systems, 7th edition, Pearson. Chapter 10.

Data Modeling: ER Model

The ER Model describes data as entities, relationships, and attributes.

- Conceptual data model
- Logical data model
- Physical data model



https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

Data Modeling: Entity Types, Entity Sets, and Keys

- An **entity type** defines a set (collection) of entities that have the same attributes (describes a schema for an entity set).
- The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**.
- An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set: a **key attribute** (underlined).

R. Elmasri and S. Navathe: Fundamentals of Database Systems, 7th edition, Pearson. Chapter 10.

Data Modeling: Relationship Types, Relationship Sets, Roles, and Structural Constraints

- A **relationship type** among multiple entity types defines a set of associations (or a **relationship set**) among entities from these entity types.
- The **degree of a relationship type** is the number of participating entity types (usually 2, called binary).
- Each entity type that participates in a relationship type plays a particular **role** in the relationship (signified by the role name).
- Relationship types usually have certain constraints that limit possible combinations of entities that may participate (e.g., cardinality ratios).

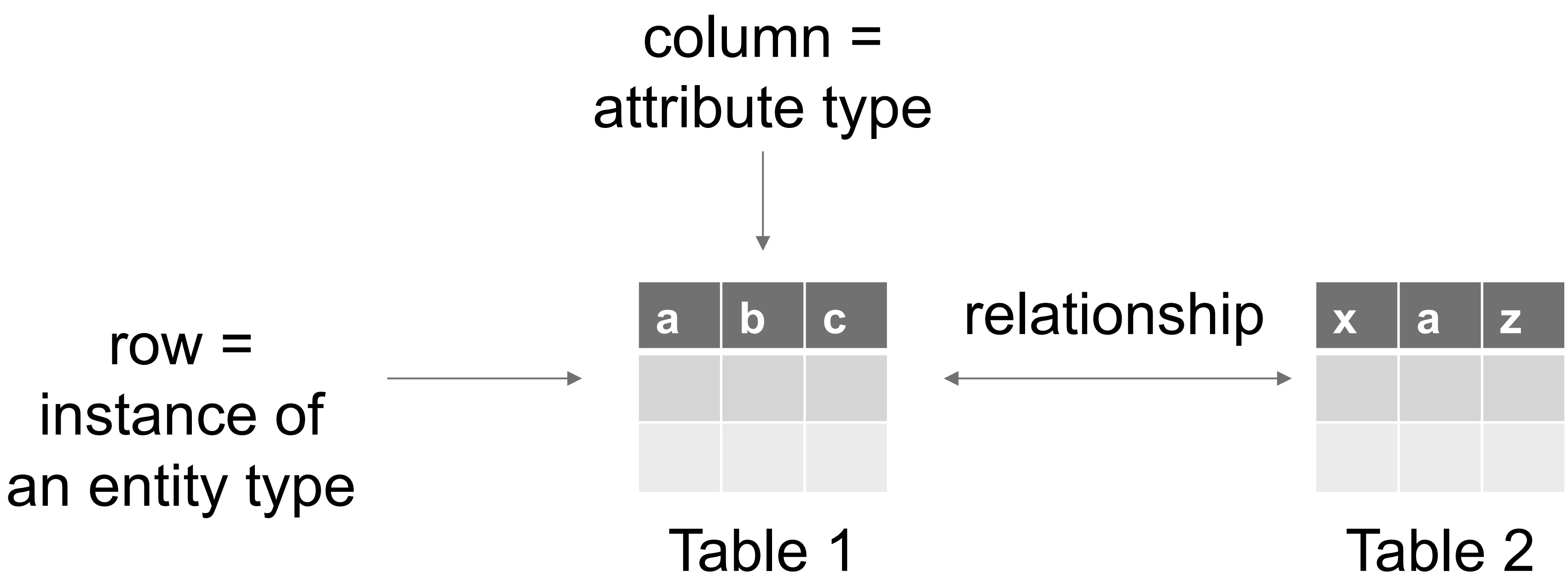
R. Elmasri and S. Navathe: Fundamentals of Database Systems, 7th edition, Pearson. Chapter 10.

Data Modeling: Weak Entity Types

- Entity types that do not have key attributes of their own are called **weak entity types**.
- Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the **identifying entity type**, and we call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.

R. Elmasri and S. Navathe: Fundamentals of Database Systems, 7th edition, Pearson. Chapter 10.

ER Model & Database Implementation



Typical Sequence of Interaction in Database Programming

The traditional architecture for database access is a three-tier client/server model:

- a top-tier client program handles display of information
- a middle-tier application program implements the business logic, including calls to one or more database servers:
    1. Open a connection (pool) to the database server
    2. Submit queries and commands (SQL statements)
    3. When database access is no longer needed, terminate the connection

R. Elmasri and S. Navathe: Fundamentals of Database Systems, 7th edition, Pearson. Chapter 10.

Object Relational Mapping (ORM)

Object Relational Mapping (ORM) is a programming technique to overcome the impedance mismatch between the relational database model and object-oriented programming language models.

For all popular object-oriented programming languages, you can find ORM libraries that enable developers to access the relational database through an object abstraction.

As and alternative or addition to ORM, the Data Access Object (DAO) design pattern can be used to bridge the object-relational impedance mismatch.

ORM with Sequelize

Sequelize is a promise-based ORM for Node.js. It supports the dialects PostgreSQL, MySQL, SQLite and MSSQL and features transaction support, relations, read replication, and more.

http://docs.sequelizejs.com/

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

ORM with Sequelize: connection setup

```
// app.js

const Sequelize = require('sequelize');

// Set up a global Postgresql DB connection pool
global.db = new Sequelize(PG_DATABASE, PG_USER, "", {
  host: PG_HOST,
  port: PG_PORT,
  dialect: 'postgres',
  define: {
    underscored: true
  }
});
```

ORM with Sequelize: data modeling

Technische Universität Berlin

```
// models/project.js

const Sequelize = require('sequelize');

module.exports = global.db.define('project', {
  title: {
    type: Sequelize.STRING
  }
}, {
  timestamps: false
});
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

ORM with Sequelize: queries and commands

```
// controllers/project.js

const Project = require('../models/project');

// find all projects
Project.findAll().then(projects => {
  console.log(projects)
}).catch(err => {
  console.error(err);
});

// create a new project
Project.create({title: "My Project"}).then(data => {
  console.log("Saved.")
}).catch(err => {
  console.error(err);
});
```

ORM with Sequelize: relationships

```
const Project = require('../models/project');
const Creator = require('../models/creator');
const Backer = require('../models/backer');
const Investment = require('../models/investment');

Creator.hasMany(Project);

Project.belongsToMany(Backer, {
  through: Investment
});

Backer.belongsToMany(Project, {
  through: Investment
});

global.db.sync();
```

ORM with Sequelize: transactions

```
// A backer makes an investment transaction
global.db.transaction(t => {
  Project.findOne(/* ... */).then(project => {
    Investment.findOne(/* ... */).then(investment => {
      project.addBacker(/* ... */).then(data => {
        Backer.findOne(/* ... */).then(backer => {
          Backer.update(/* ... */);
        });
      });
    });
  });
}).then(data => { // Transaction committed
  console.log(data);
}).catch(err => { // Transaction rolled back
  console.error(err);
});
```

Summary

- Relational databases
  - Basic database programming techniques
  - Relational database modeling using ER models
  - Programming PostgreSQL + ORM with Sequelize

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

PROGRAMMING WITH NOSQL DATABASES

Technische Universität Berlin
ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Why do we need NoSQL Databases?

- Modern Internet application requirements
  - low-latency CRUD operations
  - elastic scalability
  - high availability
  - reliable and durable storage
  - geographic distribution
  - flexible schema
- Less prioritized
  - Transactions, ACID guarantees... but some form of data consistency is still desirable
  - SQL support… but some SQL features are still desirable

NoSQL Databases: Architectures

Technische Universität Berlin

Architecture	Techniques	Systems
Dynamo-style Ring (P2P)	All nodes are equal. Each node stores a data partition + replicated data.	Cassandra, Riak, Voldemort, Amazon DynamoDB
Master-Slave	Data partitioned across slaves. Each slave stores a data partition + replicated data.	HBase, MongoDB, Redis, RethinkDB, Neo4j
Full replication	Bi-directional, incremental replication between all nodes. Each node stores all data.	CouchDB

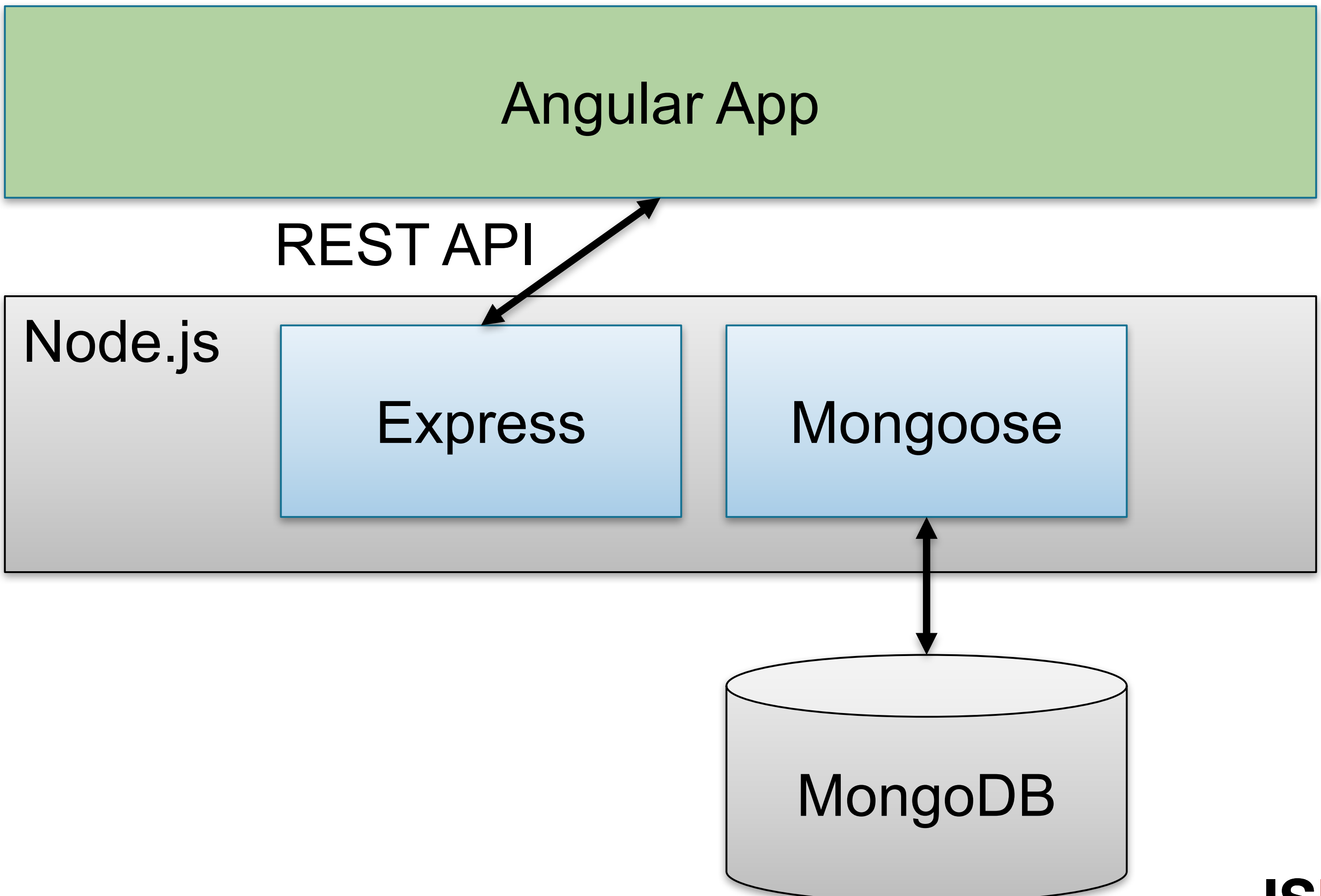
ISEngineering
Wirtschaftsinformatik – Information Systems Engineering

NoSQL Databases: Data models

Technische Universität Berlin

Data model category	Storage and Retrieval	Systems
Wide-Column	Each row stores a flexible number of columns. Data is partitioned by row key.	Cassandra, HBase, Amazon DynamoDB
Document	Storage and retrieval of structured data in the form of JSON, BSON, RDF, … documents.	CouchDB, MongoDB, RethinkDB
Key-value	Row-oriented data storage of simple (key,value) pairs in a flat namespace.	Riak, Redis, Voldemort, Yahoo! Sherpa
Graph	Storage and retrieval of data that is stored as nodes and links of graphs in a graph-space.	Neo4j

MEAN Application Architecture



ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

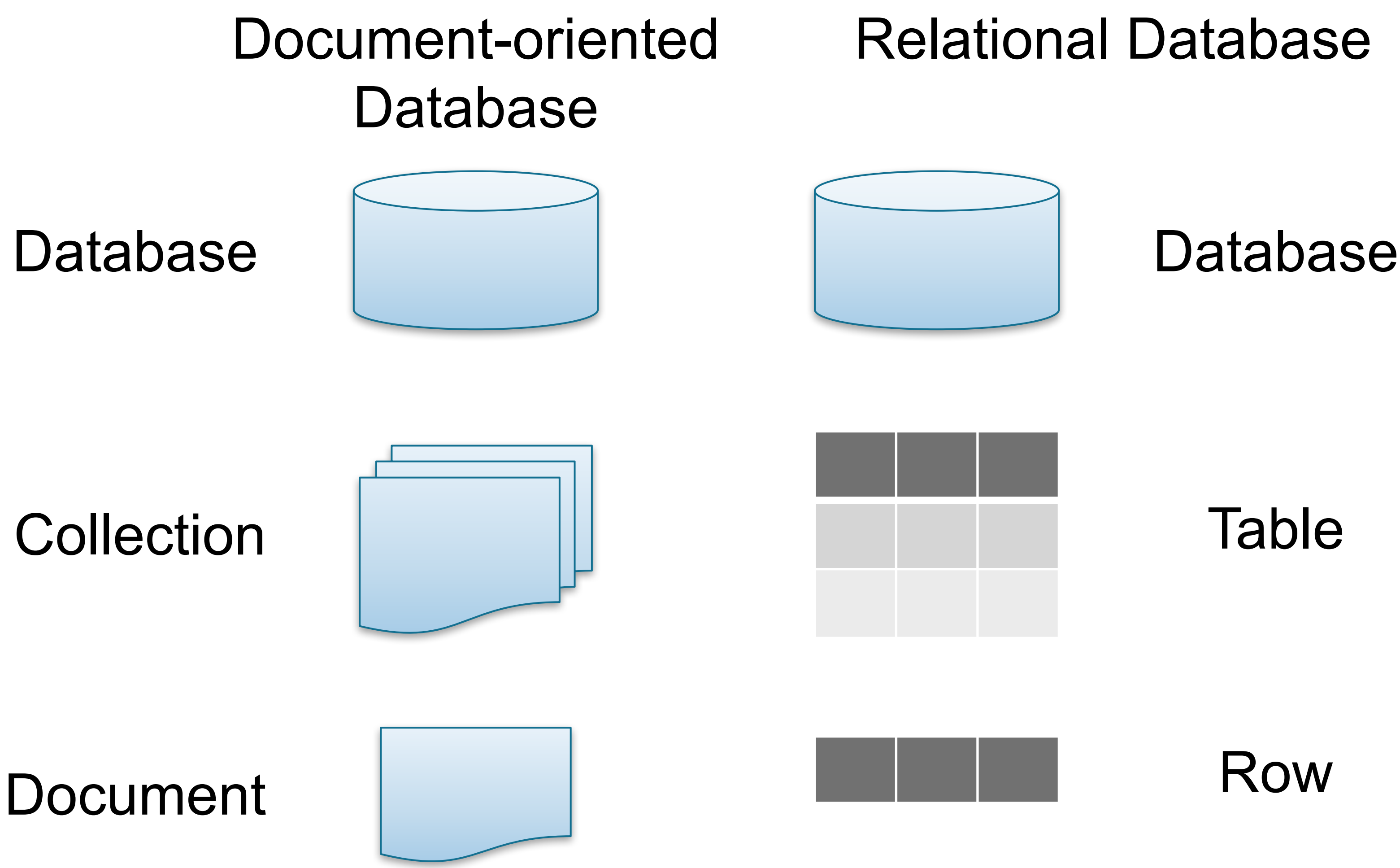
MongoDB

Technische Universität Berlin

- MongoDB is a master-slave document-oriented database system.
- Different from classical relational databases, such as MySQL or PostgreSQL, MongoDB stores data as collections of documents.
- What makes MongoDB particularly appealing to JavaScript programmers is the fact that data objects are stored in the BSON ("Binary JSON") format.

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

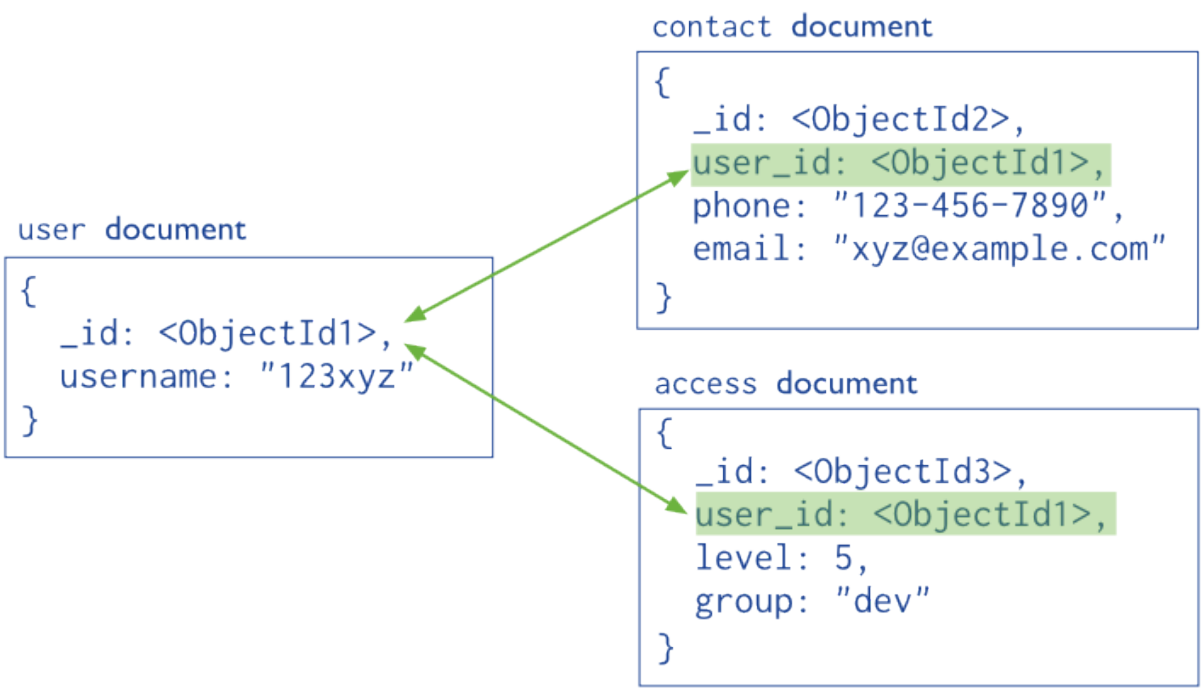
MongoDB is a document-oriented database



ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Data Modeling: Document References

- References store the relationships between data by including links or references from one document to another.
- Applications can resolve these references to access the related data.
- Broadly, these are normalized data models.



https://docs.mongodb.com/manual/core/data-modeling-introduction/

Data Modeling: Embedded Documents

- Embedded documents capture relationships between data by storing related data in a single document structure. MongoDB documents make it possible to embed document structures in a field or array within a document.
- These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.

```
{
   _id: <ObjectId1>,
   username: "123xyz",
   contact: {
               phone: "123-456-7890",
               email: "xyz@example.com"
             },
   access: {
              level: 5,
              group: "dev"
            }
}
```

Embedded sub-document

Embedded sub-document

https://docs.mongodb.com/manual/core/data-modeling-introduction/

Data Modeling

Use References when

- embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication
- to represent more complex many-to-many relationships
- to model large hierarchical data sets.

Use Embedded Documents when

- you have “contains” relationships between entities
- you have one-to-many relationships between entities

https://docs.mongodb.com/manual/core/data-modeling-introduction/

MongoDB CRUD Operations: Create

https://docs.mongodb.com/manual/crud/

MongoDB CRUD Operations: Read

```
db.users.find(                          <- collection
   { age: { $gt: 18 } },                <- query criteria
   { name: 1, address: 1 }              <- projection
).limit(5)                              <- cursor modifier
```

https://docs.mongodb.com/manual/crud/

MongoDB CRUD Operations: Update

```
db.users.update(                                   ← collection
   { age: { $gt: 18 } },                           ← update criteria
   { $set: { status: "A" } },                      ← update action
   { multi: true }                                 ← update option
)
```

https://docs.mongodb.com/manual/crud/

MongoDB CRUD Operations: Delete

```
db.users.remove(          ← collection
   { status: "D" }        ← remove criteria
)
```

https://docs.mongodb.com/manual/crud/

MongoDB by Example: First Steps

When connected to the mongo shell, show the dbs. There should only be an empty local db.

```
$ mongo
show dbs
local  0.000GB
```

Create and switch to a database named dev as follows:

```
$ mongo
use dev
switched to db dev
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Writing data

- Insert a single document with 3 properties into a collection, named `hotels`.
- If the collection does not exist, yet, it will be automatically created when the `insert` operation is executed.

```
$ mongo
```

```
db.hotels.insert({
    "name": "Vulcan Inn",
    "stars": 2,
    "planet": "Vulcan"
})
```

```
WriteResult({ "nInserted" : 1 })
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Writing data

Technische Universität Berlin

- Insert another, more complex document into the `hotels` collection.

```
$ mongo
```

```
db.hotels.insert({
    "name": "Deep Space 9 Lodges",
    "stars": 3,
    "tags": ["wormhole", "Terok Nor", "DS9"],
    "armament": {
        "phaser arrays": 4,
        "torpedo launchers": 10
    }
})
```

```
WriteResult({ "nInserted" : 1 })
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Writing data

```
$ mongo
```

```
db.hotels.insertMany([{
   name: 'Star Fleet Motel',
   stars: 3
}, {
   name: 'Romulus Resorts',
   stars: 5
}, {
   name: 'Klingon BnB',
   stars: 1
}])
```

```
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("57aafe7ecc265c6cc5a874c3"),
                ObjectId("57aafe7ecc265c6cc5a874c4"),
                ObjectId("57aafe7ecc265c6cc5a874c5")
        ]
}
```

MongoDB: Reading data

- Query all documents that are in the `hotels` collection via `db.hotels.find({})` or, shorter, via `db.hotels.find()`.

```
$ mongo
```

```
db.hotels.find()
```

```
{ "_id" : ..., "name" : "Vulcan Inn", "stars" : 2, "planet" : "Vulcan" }
{ "_id" : ..., "name" : "Star Fleet Motel", "stars" : 3 }
{ "_id" : ..., "name" : "Romulus Resorts", "stars" : 5 }
{ "_id" : ..., "name" : "Klingon BnB", "stars" : 1 }
{ "_id" : ..., "name" : "Deep Space 9 Lodges", "stars" : 3, ... }
```

- A you can see, the insert operations have created objects with an auto-generated `_id` attribute.
- You can insert documents with your own `_id` attribute by explicitly stating the id, for example: `db.foo.insert({"_id": 3,"bar": "baz"})`

MongoDB: Reading data

Technische Universität Berlin

- Query documents that match the "name" field:

```
$ mongo
```

```
db.hotels.find({
    "name": "Klingon BnB"
})
```

```
{ "_id" : ..., "name" : "Klingon BnB", "stars" : 1 }
```

- Query documents that match the "stars" field:

```
$ mongo
```

```
db.hotels.find({
    "stars": 3
})
```

```
{ "_id" : ..., "name" : "Star Fleet Motel", "stars" : 3 }
{ "_id" : ..., "name" : "Deep Space 9 Lodges", "stars" : 3,
... }
```


ISEngineering
Wirtschaftsinformatik – Information Systems Engineering

MongoDB: Reading data

- Query a document that matches both `"name"` AND `"stars"` fields.

```
$ mongo
```

```
db.hotels.find({
    "name": "Star Fleet Motel",
    "stars": 3
})
```

```
{ "_id" : ..., "name" : "Star Fleet Motel", "stars" : 3 }
```

MongoDB: Reading data

- Query documents that contain a matching item inside the `"tags"` array.

```
$ mongo
```

```
db.hotels.find({
    "tags": "DS9"
})
```

```
{ "_id" : ..., "name" : "Deep Space 9 Lodges", "stars" : 3,
"tags" : [ "wormhole", "Terok Nor", "DS9" ], ... }
```

MongoDB: Reading data

- Query documents with a “less-than” query selector, to find `hotels` with less than 3 stars.

```
$ mongo
```

```
db.hotels.find({
   "stars": {
       "$lt": 3
   }
})
```

```
{ "_id" : ..., "name" : "Vulcan Inn", "stars" : 2, "planet" : "Vulcan" }
{ "_id" : ..., "name" : "Klingon BnB", "stars" : 1 }
```

MongoDB: Updating data

- The Klingon BnB has upgraded its amenities.
- Update (set) the `"stars"` attribute of the Klingon BnB to 2 stars.
- The first JSON object is the query parameter and the second object is the update parameter.

```
$ mongo
```

```
db.hotels.update(
    // first, the QUERY parameter
    {
        "name": "Klingon BnB"
    },
    // second, the UPDATE parameter
    {
        "$set": { "stars": 2 }
    }
)
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1
})
```

MongoDB: Updating data

- Add an additional field with the name `"type"` and with the value `"hotel"` to all documents in the collection.

```
$ mongo
```

```
db.hotels.update(
    // QUERY parameter: find all documents
    {},
    // UPDATE parameter: add the "type" attribute
    { "$set": { "type": "hotel" } },
    // update ALL matching documents,
    // otherwise only the first match is updated
    { "multi": true }
)
```

```
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4
})
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Updating data

- Update the first item (with index 0) in the `"tags"` array of `"Deep Space 9 Lodges"` from `"wormhole"` to `"Bajoran wormhole"`.

```
$ mongo
```

```
db.hotels.update(
   // QUERY parameter
   {
       "name": "Deep Space 9 Lodges"
   },
   // UPDATE parameter
   { "$set": { "tags.0": "Bajoran wormhole" } }
)
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1
})
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Increment numeric attributes

- We can perform an atomic increment (or decrement) of numeric attributes with the `$inc` operator
- Note the difference between `$set` and `$inc` operators: the `$set` operator sets a particular value, the `$inc` operator increments a numeric value by another number.

```
$ mongo
```

```
db.hotels.update(
    // QUERY parameter
    {
        "name": "Vulcan Inn",
        "type": "hotel"
    },
    // INCREMENT parameter
    { "$inc": { "stars": 2 } }
)
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

MongoDB: Increment numeric attributes

- Unfortunately, drunken Klingons destroyed the nice amenities of the Klingon BnB.
- Decrement the stars rating of the Klingon BnB by 1 star (decrementing is just incrementing with negative numbers).

```
$ mongo
```

```
db.hotels.update(
    // QUERY parameter
    {
        "name": "Klingon BnB",
        "type": "hotel"
    },
    // INCREMENT parameter
    { "$inc": { "stars": -1 } }
)
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

MongoDB: Renaming attributes

- We have trademarked our five-star™ review method. This requires a renaming of our field from `"stars"` to `"stars_tm"`.

```
$ mongo
```

```
db.hotels.update(
   // QUERY parameter
   {
       "type": "hotel"
   },
   // RENAME parameter
   {
       "$rename": { "stars": "stars_tm" }
   },
   // update ALL matching documents
   { "multi": true }
)
```

```
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })
```

MongoDB: Removing data

- Remove the `"Star Fleet Motel"` document:

```
$ mongo
```

```
db.hotels.remove({
    "name": "Star Fleet Motel"
})
```

```
WriteResult({ "nRemoved" : 1 })
```

MongoDB: Removing data

- Remove the `"planet"` field from the Vulcan Inn.
- The value that we provide to the field which we `$unset` does not matter, as shown in the example below with `"val_doesnt_matter"`.

```
$ mongo
```

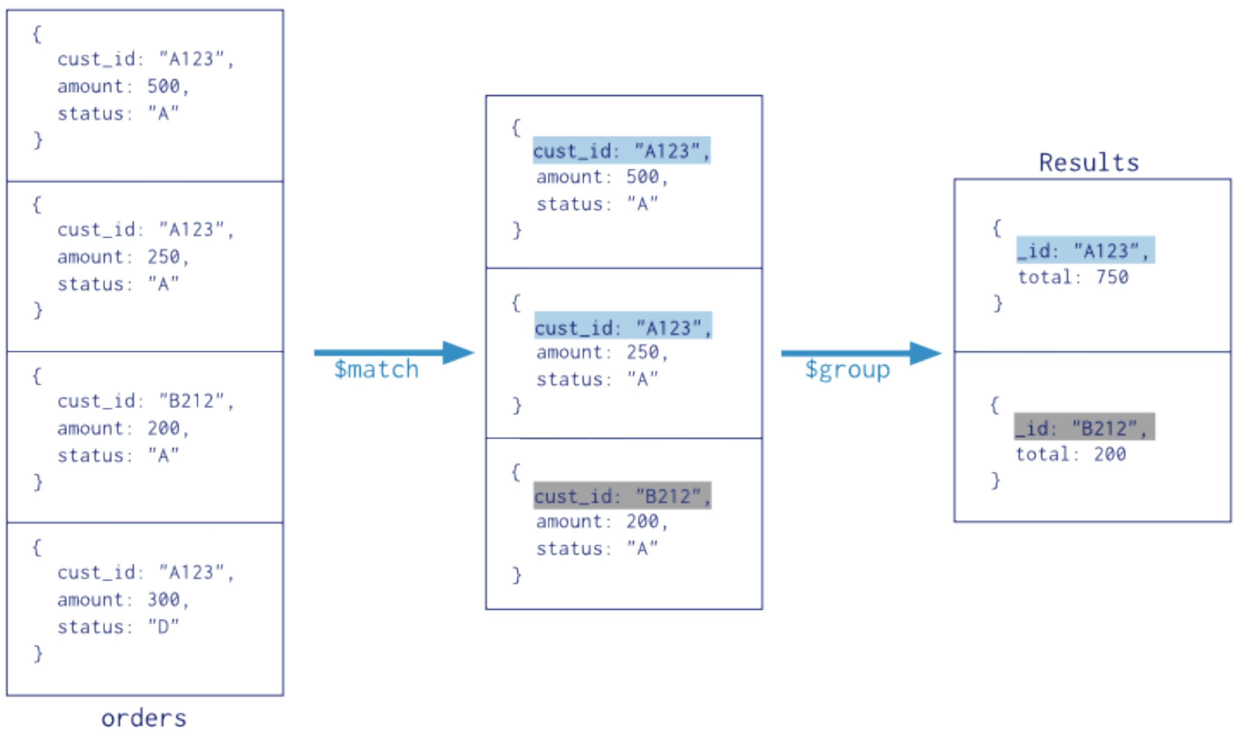
```
db.hotels.update(
   // QUERY parameter
   {
      "name": "Vulcan Inn"
   },
   // UNSET parameter
   {
      "$unset": {
         "planet": "val_doesnt_matter"
      }
   }
)
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

MongoDB: Aggregations

Collection

```
db.orders.aggregate( [
    $match stage →   { $match: { status: "A" } },
    $group stage →   { $group: { _id: "$cust_id",total: { $sum: "$amount" } } }
                 ] )
```



https://docs.mongodb.com/manual/aggregation/

ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Aggregations

- Create an aggregation by hotel `$name`.

```
$ mongo
```

```
db.hotels.aggregate({
   "$group": {
       "_id": "$name"
   }
})
```

```
{ "_id" : "Vulcan Inn" }
{ "_id" : "Deep Space 9 Lodges" }
{ "_id" : "Klingon BnB" }
{ "_id" : "Romulus Resorts" }
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Aggregations

- We insert some more documents, to perform more interesting aggregations.

```
$ mongo
```

```
db.hotels.insertMany([{
    "name": "Starfleet Hotel",
    "type": "hotel",
    "location": "Bahamas",
    "stars_tm": 5
}, {
    "name": "Starfleet Hotel",
    "type": "hotel",
    "location": "Frankfurt",
    "stars_tm": 3
}])
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

MongoDB: Aggregations

- Now, create an aggregation with group key name and accumulate average `$stars_tm` as `"avg_stars"`.
- Note that the Starfleet Hotel's rating is 4, which is, indeed, the average of the two values 5 and 3.

```
$ mongo
```

```
db.hotels.aggregate({
   "$group": {
       "_id": "$name",
       "avg_stars": {
           "$avg": "$stars_tm"
       }
   }
})
```

```
{ "_id" : "Starfleet Hotel", "avg_stars" : 4 }
{ "_id" : "Vulcan Inn", "avg_stars" : 4 }
{ "_id" : "Deep Space 9 Lodges", "avg_stars" : 3 }
{ "_id" : "Klingon BnB", "avg_stars" : 1 }
{ "_id" : "Romulus Resorts", "avg_stars" : 5 }
```

Connecting Node.js to MongoDB

- Connecting mongodb with node.js
  - Install the mongodb driver for node.js via npm
  - Connect to a database by calling the MongoClient.connect() method and pass the URI of the mongodb database, and a callback function that either returns an error or a database handle object
  - Now you can insert and query dataNon-blocking I/O via callbacks

- In the following, we will write a simple node.js application with express routing that connects to mongodb for reading and writing persistent data.

Connecting Node.js to MongoDB: Example

Technische Universität Berlin

index.js

```
// MongoDB connection setup
const mongo = require('mongodb').MongoClient;
const host = process.env.MONGO_PORT_27017_TCP_ADDR;
const port = process.env.MONGO_PORT_27017_TCP_PORT;
const mongodbURL = 'mongodb://' + host +  ':' + port + '/dev';

let db;
mongo.connect(mongodbURL, (err, database) => {
 if (err) {
   console.log(err);
 } else {
   db = database;

   // Launch the application
   app.listen(3000, () => console.log('Started'));
   console.log('It works!');
 }
});
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Connecting Node.js to MongoDB: Example (2)

- Let us now add `express` and a first `express route` that retrieves all documents in the `hotels` collection from mongodb.

index.js

```
// database setup ...
const express = require('express');
const app = express();

// Get a list of hotels
app.get('/hotels', (req, res) => {
 let cursor = db.collection('hotels').find();
 cursor.toArray((err, docs) => {
   if (err) {
     console.log(err);
   }
   res.send(docs);
 });
});
```

Connecting Node.js to MongoDB: Example (3)

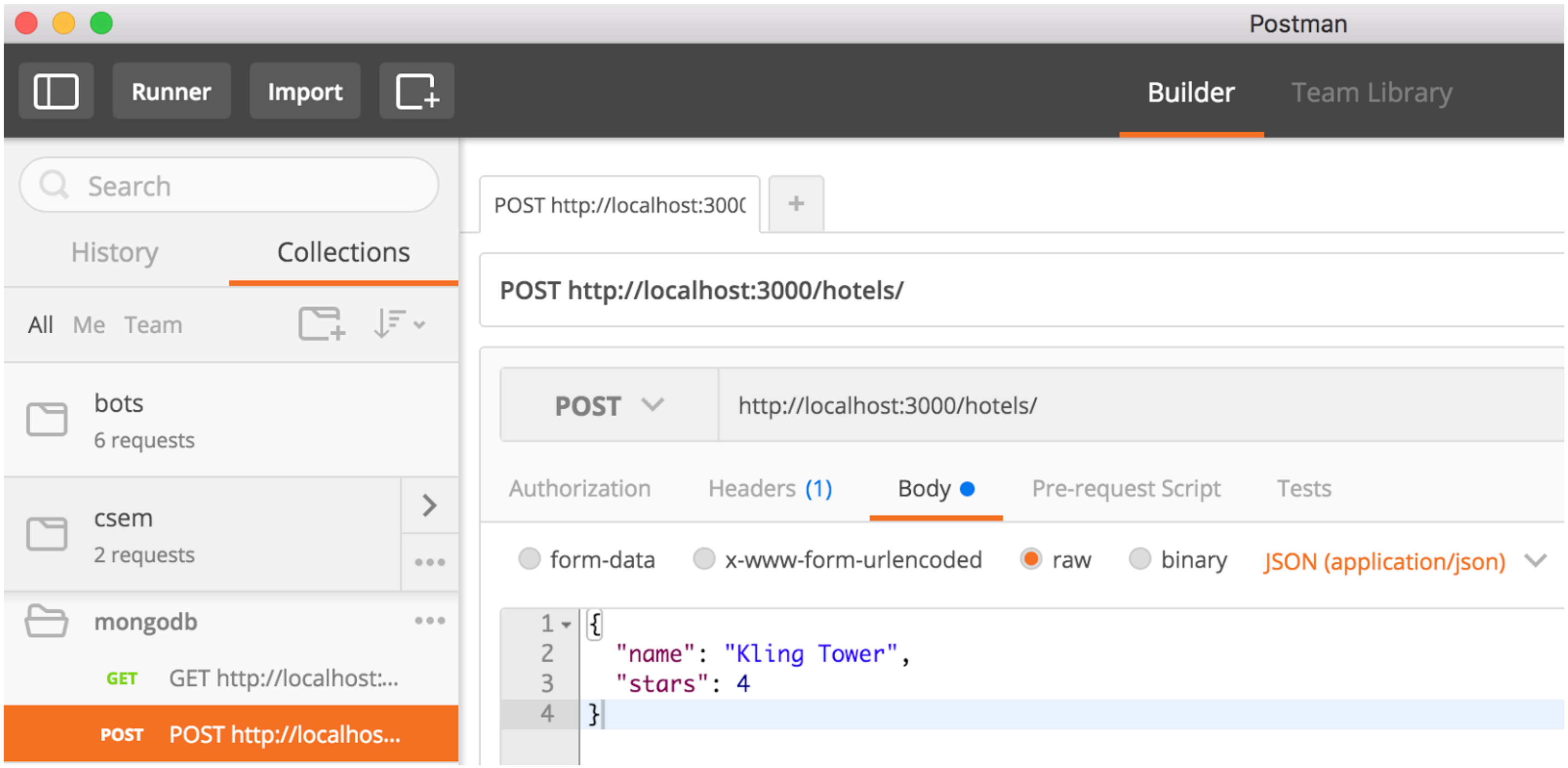
Technische Universität Berlin

index.js

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');

// other code …

app.use(bodyParser.json()); // for parsing application/json

// Create a new hotel
app.post('/hotels', (req, res) => {
 db.collection('hotels').save(req.body, (err, docs) => {
   if (err) {
     console.log(err);
   }
   console.log('saved a new hotel');
   res.redirect('/hotels');
 });
});
```

Connecting Node.js to MongoDB: Example (4)

Technische Universität Berlin



ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

ODM with Mongoose

- We connect Node.js and MongoDB via Object-Document Mapping (ODM).
- The most popular mapper for mongodb is mongoose.
    - Mongoose provides schema validation, pseudo joins, and other features for developers.
    - The mongoose API consists of 4 main data types:
        - **Schema:** specifies rules that a valid document must satisfy, documents can only be inserted into a collection if they satisfy the schema's constraints
        - **Connection:** consists of one or more sockets to a mongodb database server
        - **Model:** Associated with both schema and connection, model wraps around a single mongodb document or collection
        - **Document:** A mongoose document is the instantiation of a single model

ODM with Mongoose: Example

Technische Universität Berlin

- For using mongoose, we need to connect our mongoose object (instead of the mongodb object) with the MongoDB database.

index.js

```
// MongoDB connection setup
const mongodb = require('mongodb');
const mongoose = require('mongoose');
const host = process.env.MONGO_PORT_27017_TCP_ADDR;
const port = process.env.MONGO_PORT_27017_TCP_PORT;
const mongodbURL = 'mongodb://' + host +  ':' + port + '/dev';

mongoose.connect(mongodbURL);
let db = mongoose.connection;
db.on('error', console.error.bind(console, 'conn error:'));
db.once('open', () => {
 app.listen(3000, () => console.log(’Connect via mongoose'));
 console.log('It works!');
});
```

ODM with Mongoose: Example (2)

- First and foremost, the mongoose ODM allows us to separate schema definitions from the rest of our code.
- Let’s create a schema for our hotel documents and put it in a models subdirectory.

models/hotel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

module.exports = new Schema({
 name:  String,
 stars: Number,
});
```

ODM with Mongoose: Example (3)

- Now we can import the hotel schema into our main application code.

index.js

```
// connect to MongoDB via mongoose ...

// Schemas and models
const hotelSchema = require('./models/hotel.js');
let Hotel = mongoose.model('Hotel', hotelSchema, 'hotels');
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

ODM with Mongoose: Example (4)

index.js

```
// other code ...

// Get a list of hotels
app.get('/hotels', (req, res) => {
 Hotel.find().exec(function (err, hotels) {
      if (err) {
        console.log(err);
        res.status(400).send('Whoops. An error occured.');
      } else {
        res.send(hotels);
      }
    });
});
```

ODM with Mongoose: Example (5)

index.js

```
// other code ...

// Create a new hotel
app.post('/hotels', (req, res) => {
 let h = new Hotel(req.body);
 h.save((err, h) => {
   if (err) {
     console.log(err);
     res.status(400).send(err);
   } else {
     res.redirect('/hotels');
   }
 });
});
```

Mongoose Validators

- A main benefit of mongoose are so-called middleware functions (also called pre- and post-hooks) which are similar to express middleware functions that we discussed in a previous lecture.
- A commonly needed type of middleware function are validators.
    - Validators are by default built-in as pre(‘save’) hooks or can, alternatively, be called manually.
    - A validator checks if a model complies with the defined Schema.

Mongoose Validators: Example

models/hotel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;


module.exports = new Schema({
 name:  {
   type: String,
   required: true,
 },
 stars: Number,
});
```

Mongoose Validators: Example (2)

- If we now send an HTTP POST request with body { stars: 4 } at the endpoint http://localhost:3000/hotels/, we get the following error message:


```
{
 "message": "Hotel validation failed",
 "name": "ValidationError",
 "errors": {
   "name": {
     "message": "Path `name` is required.",
     "name": "ValidatorError",
     "properties": {
       "type": "required",
       "message": "Path `{PATH}` is required.",
       "path": "name"
     },
     "kind": "required",
     "path": "name"
   }
 }
}
```


ISEngineering
Wirtschaftsinformatik –
Information Systems Engineering

Mongoose Validators: Example (3)

Technische Universität Berlin

- If a property type is wrong and cannot be casted, we get another error. For example, let's send an HTTP POST request with the following body:

```
{
 "name": "Luxury Resorts",
 "stars": "five"
}
```

- Then, we get a validation error that says:

```
{
 "message": "Hotel validation failed",
 "name": "ValidationError",
 "errors": {
   "stars": {
     "message": "Cast to Number failed for value \"five\"
                at path \"stars\"", ...
```


ISEngineering
Wirtschaftsinformatik – Information Systems Engineering

Summary

Technische Universität Berlin

- Document-oriented databases
  - Basic modeling techniques: references & embedding
  - MongoDB queries & commands
  - ODM with mongoose