
LUT
University
LAND
OF THE
CURIOUS

TABLE OF CONTENTS

CT60A4304 - BASICS OF DATABASE SYSTEMS

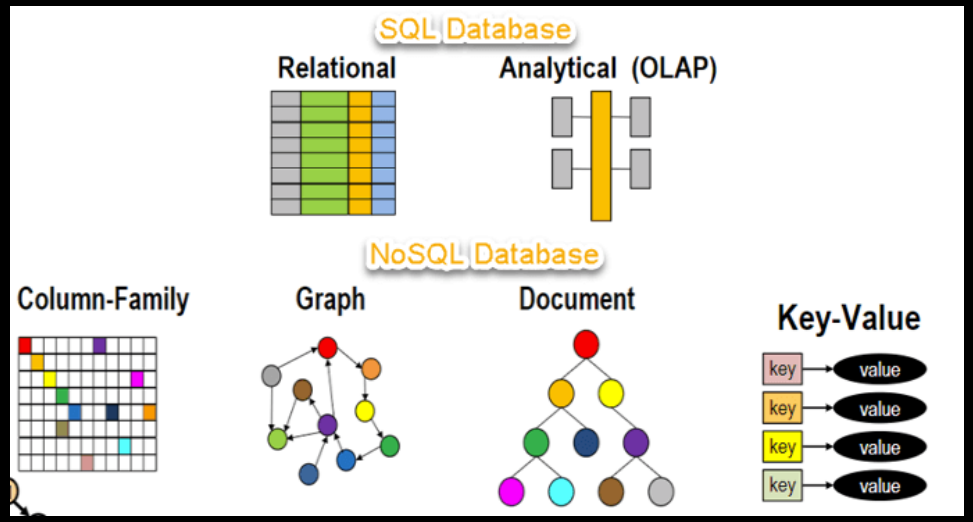
NOSQL DATABASES

Lecture

Jiri Musto, D.Sc.

NOSQL DATABASES

- Stands for “Not Only SQL”
- There are no fixed schemas
- Avoids joins
- Used for big data and real-time web apps
- Can be structured, semi-structured or unstructured data
- Developed in the 2000s



BENEFITS AND DRAWBACKS

- Benefits
  - Flexible
  - Scale-Out (distributed system)
  - Fast-paced development (no need for detailed modelling)
  - Semi-structured data
- Drawbacks
  - Relational database is better for scaling up (one system, more RAM, CPU, HDD)
  - Inconsistencies (not good for transaction oriented systems like bank)
  - Less secure
  - Lack of standardization

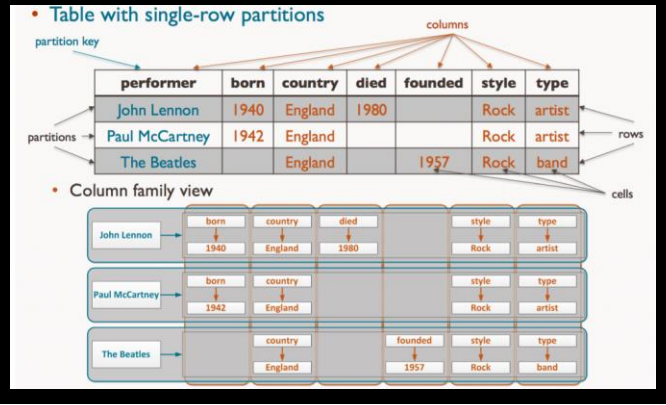
RELATIONAL VS. NOSQL

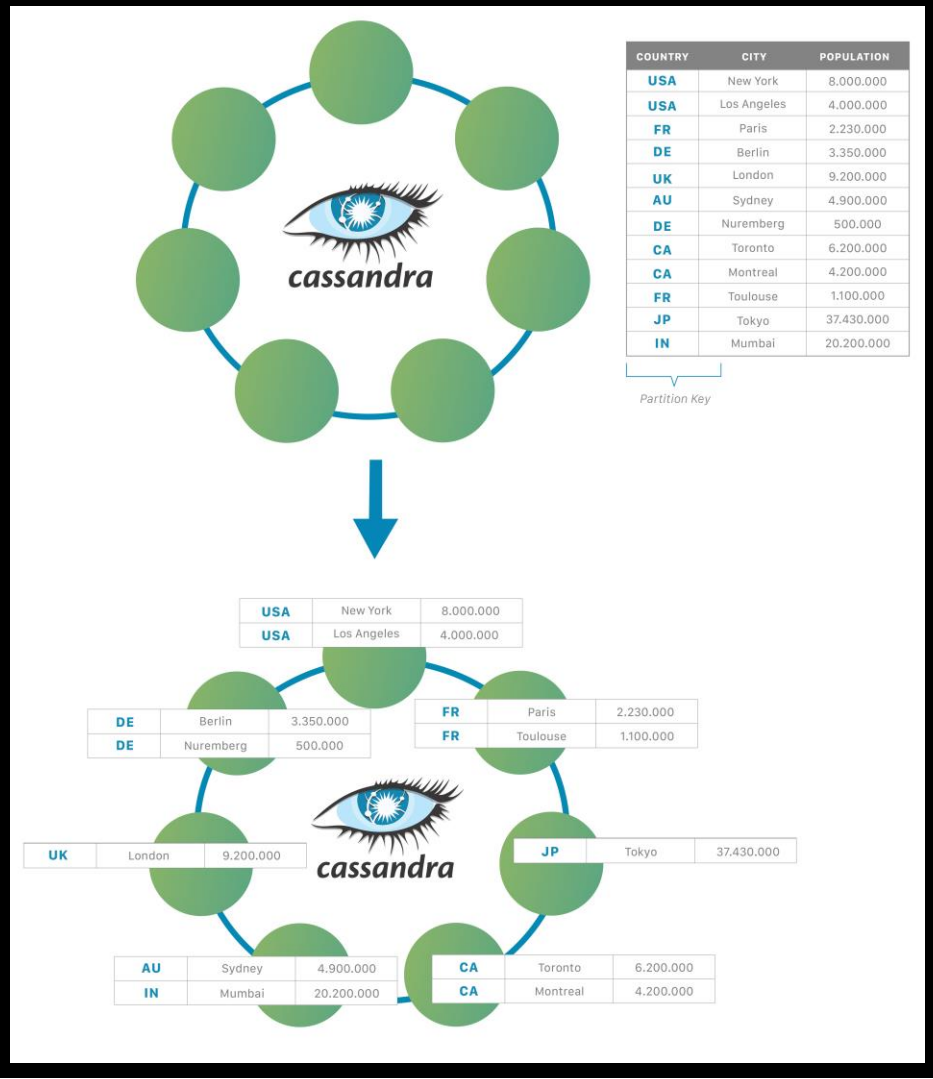
Relational	NoSQL
Predefined schema	Schemaless
Transactions, consistent	Eventual consistency
Scale-up	Scale-out
Standard SQL language	No real standardization
Easy record retrieval	Easier to store large amounts of data
Structured data	Semi-structured data (or even unstructured)
Good for complex queries	Good for hierarchical data
Normalized schema	De-normalized schema

https://www.softwaretestinghelp.com/sql-vs-nosql/

COLUMN DATABASE

- Useful in analytical applications or recommendation engines
- Columns are treated separately
- Can access rows with a “row key”
- Example: Apache Cassandra and Apache Hbase
- Use cases: Facebook, Spotify
- Main benefit is partitioning data and aggregation functions





GRAPH DATABASE

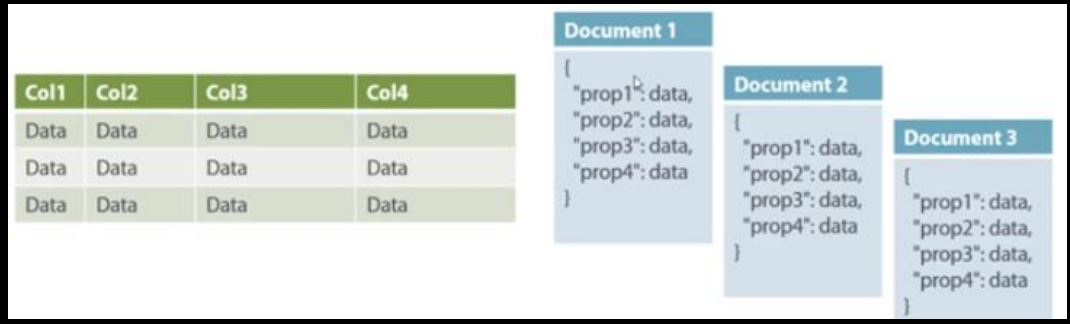
- Stores the relationships between data as nodes
- No need to join and match data from separate tables
- Useful in social networks, product recommendations, access management, and similar scenarios
- Example: Neo4j
- Use cases: Walmart, Cisco

KEY-VALUE DATABASE

- Similar to Python dictionary, associative arrays, maps, hash, or other data structures
- Each data has a key that directs to it
- Value can be of any type (even data structures)
- Used in session management and caching in web applications (login systems), user profiles, blog comments, e-commerce
- Example: Redis, Dynamo
- Use cases: Twitter timeline, Pinterest, Quora

DOCUMENT DATABASE

- Uses JSON, XML or BSON documents for storage
- No predefined structure, documents can be different
- Documents can have sub-documents or reference other documents
- Accessing documents and fields within document works similar to key-value pairing
- Can have their own query language as well as a universal API system
- Useful in content management systems, e-commerce and real-time analytics
- Example: CouchDB, MongoDB
- Use cases: Twitter, Google, ebay



EXAMPLE OF DOCUMENT DATABASE


```
{
    "_id": {
        "$oid": "620b26403845b9c5c7042819"
    },
    "uid": 84001001,
    "country_iso2": "US",
    "country_iso3": "USA",
    "country_code": 840,
    "fips": 1001,
    "county": "Autauga",
    "state": "Alabama",
    "country": "US",
    "combined_name": "Autauga, Alabama, US",
    "population": 55869,
    "loc": {
        "type": "Point",
        "coordinates": [-86.6441, 32.5395]
    },
    "date": {
        "$date": "2020-01-22T00:00:00.000Z"
    },
    "confirmed": 0,
    "deaths": 0,
    "confirmed_daily": 0,
    "deaths_daily": 0
}
```

```
{
    "_id": {
        "$oid": "620b26403845b9c5c704281a"
    },
    "uid": 84001001,
    "country_iso2": "US",
    "country_iso3": "USA",
    "country_code": 840,
    "fips": 1001,
    "county": "Autauga",
    "state": "Alabama",
    "country": "US",
    "combined_name": "Autauga, Alabama, US",
    "population": 55869,
    "loc": {...},
    "date": {...},
    "confirmed": 0,
    "deaths": 0,
    "confirmed_daily": 0,
    "deaths_daily": 0
}
```

```
_id: ObjectId("5e7a6f9f1f27c900177b30ea")
userId: ObjectId("5e7a6f9f1f27c900177b30eb")
prevVal: Object
    initialize: "New report"
newVal: Object
    initialize: "Report initialized"
systemTimeStamp: "Tue Mar 24 2020 20:37:51 GMT+0000 (Coordinated Universal Time)"
__v: 0
```

```
_id: ObjectId("5e7a70a21f27c900177b30ef")
additionalInfo: "You can use mobile phones to submit as well."
photo: "http://res.cloudinary.com/dtcshxdss/raw/upload/v1585082531/ffneygasssx..."
objectivityOfPath: 0
accuracyOfPath: 0
pathPhysicalPropertiesId: ObjectId("5e7a70a21f27c900177b30f1")
pathEnvironmentalPropertiesId: ObjectId("5e7a70a21f27c900177b30f0")
__v: 0
```

```
[{
  "_id": {
    "$oid": "5ea088738834618b60b586d1"
  },
  "created_at": "Wed Apr 22 18:09:11 +0000 2020",
  "id": 1253023058413129700,
  "id_str": "1253023058413129730",
  "text": "@BobHoldenNYC Its where the corona breeds",
  "truncated": false,
  "entities": {
    "hashtags": [],
    "symbols": [],
    "user_mentions": [
      {
        "screen_name": "BobHoldenNYC",
        "name": "Robert Holden",
        "id": 948549529615392800,
        "id_str": "948549529615392769",
        "indices": [
          0,
          13
        ]
      }
    ],
    "urls": []
  },
  "metadata": {
    "iso_language_code": "en",
    "result_type": "recent"
  },
  "source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter for Android</a>",
  "in_reply_to_status_id": 1253012337667309600,
  "in_reply_to_status_id_str": "1253012337667309574",
  "in_reply_to_user_id": 948549529615392800,
  "in_reply_to_user_id_str": "948549529615392769",
  "in_reply_to_screen_name": "BobHoldenNYC",
  "user": {
    "id": 1042728872,
    "id_str": "1042728872",
    "name": "Rexxx",
    "screen_name": "Rexxx0001",
    "location": "New York, USA",
    "description": "Hard rock/metal/drummer/scoundrel /animal lover/Yankee fan/gamer. #TheLucasOrder",
    "url": null,
    "entities": {
      "description": {
        "urls": []
      }
    },
    "protected": false,
    "followers_count": 115,
    "friends_count": 331,
    "listed_count": 1,
    "created_at": "Fri Dec 28 17:32:20 +0000 2012",
    "favourites_count": 14182,
    "utc_offset": null,
    "time_zone": null,
    "geo_enabled": false,
    "verified": false,
    "statuses_count": 3350,
    "lang": null,
    "contributors_enabled": false,
    "is_translator": false,
    "is_translation_enabled": false,
    "profile_background_color": "C0DEED",
    "profile_background_image_url": "http://abs.twimg.com/images/themes/theme1/bg.png",
    "profile_background_image_url_https": "https://abs.twimg.com/images/themes/theme1/bg.png",
    "profile_background_tile": false,
    "profile_image_url": "http://pbs.twimg.com/profile_images/1219784454941691904/UjMT6N8y_normal.jpg",
    "profile_image_url_https": "https://pbs.twimg.com/profile_images/1219784454941691904/UjMT6N8y_normal.jpg",
    "profile_banner_url": "https://pbs.twimg.com/profile_banners/1042728872/1570210305",
    "profile_link_color": "1DA1F2",
    "profile_sidebar_border_color": "C0DEED",
    "profile_sidebar_fill_color": "DDEEF6",
    "profile_text_color": "333333",
    "profile_use_background_image": true,
    "has_extended_profile": false,
    "default_profile": true,
    "default_profile_image": false,
    "following": null,
    "follow_request_sent": null,
    "notifications": null,
    "translator_type": "none"
  },
  "geo": null,
  "coordinates": null,
  "place": null,
  "contributors": null,
  "is_quote_status": false,
  "retweet_count": 0,
  "favorite_count": 0,
  "favorited": false,
  "retweeted": false,
  "lang": "en"
},{
```


This is one tweet from Twitter database

USING MONGODB IN A PROGRAM: EASY EXAMPLE

- Setup the database connection
  - Port, address, username, client
- Connect to database
- Insert into database
- Read from database
  - Within the program
  - Or with external DBMS
    - MongoDB compass
    - MongoDB shell

```
//Port Setup....  ========================================
const PORT = process.env.PORT || 5000;
//Uses ===================================================
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, "client", "build")));

var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/";

function saveToDB(myobj, collectionName) {
    MongoClient.connect(url, function(err, db) {
        if (err) throw err;
        var dbo = db.db("moreTweets");
        dbo.collection(collectionName).insertOne(myobj, function(err, res) {
          if (err) throw err;
          db.close();
        });
      });
}

function tweetsFromNewYork(question, numberOfTweets) {
    console.log("Twittering!!!");
    clientApp.get(' https://api.twitter.com/1.1/search/tweets.json',
    {
        q : question,
        geocode: "40.663471,-73.137798,75mi",
        result_type :"recent",
        lang: "en",
        count: numberOfTweets
    },
    function(error, tweets, response) {
      if(error) throw error;
      tweets.statuses.forEach(element => {
          saveToDB(element, "newYorkTweets");
      });
      console.log("Tweets saved!");
    });
}
```

USING A SCHEMA/MODEL WITH MONGODB

- Middleware / library called ***mongoose***
- Make objects from JSON collections
- Can assign data types, default values, restrictions, etc
  - Similar to integrity constraints in SQL
- Can also assign functions to mongoose models
  - Similar to object-oriented programming

```
// grab the mongoose module
var mongoose = require('mongoose');

// define our address model
// module.exports allows us to pass this to oth
module.exports = mongoose.model('Address',
{
    street : {
        type : String,
        default: '',
        required: true
    },
    city : {
        type : String,
        default: ''
    },
    geoLocationId : {
        type : mongoose.ObjectId,
        require: true
    },
    precisionOfAddress : {
        type : String,
        default: '0'
    },
    accuracyOfLocation : {
        type : String,
        default: '0'
    },
    completenessOfLocation : {
        type : Number,
        default: 0
    }

}, 'Address');
```

```
const kittySchema = new mongoose.Schema({
  name: String
});

const Kitten = mongoose.model('Kitten', kittySchema);

const silence = new Kitten({ name: 'Silence' });
console.log(silence.name); // 'Silence'

// NOTE: methods must be added to the schema before compiling it with mongoose.model()
kittySchema.methods.speak = function speak() {
  const greeting = this.name
    ? "Meow name is " + this.name
    : "I don't have a name";
  console.log(greeting);
};

const Kitten = mongoose.model('Kitten', kittySchema);

const fluffy = new Kitten({ name: 'fluffy' });
fluffy.speak(); // "Meow name is fluffy"
```

CT60A4304 - BASICS OF DATABASE SYSTEMS

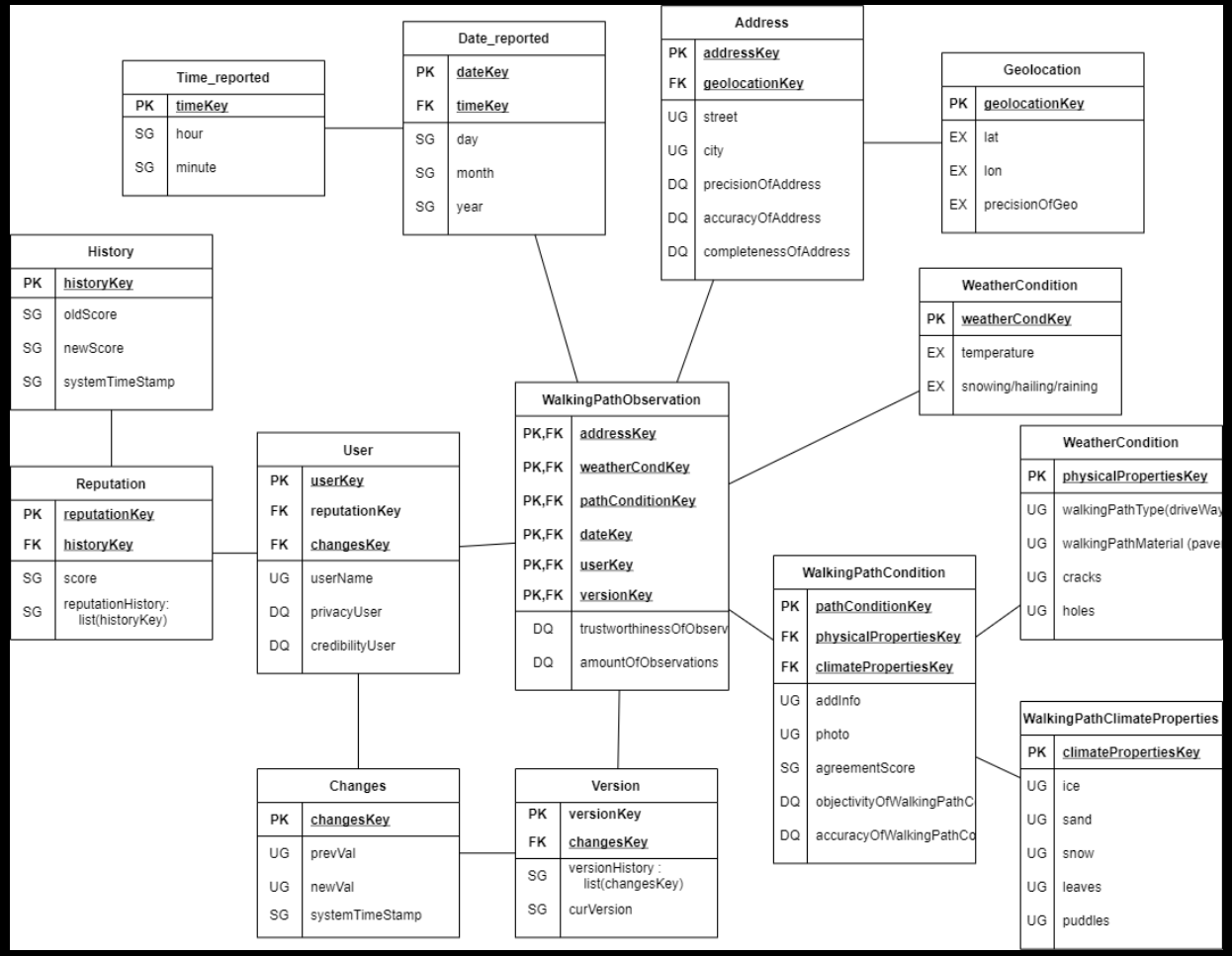
MONGODB USE CASE

Lecture

Jiri Musto, D.Sc.

DATABASE SCHEMA

- Modeled as a snowflake schema
- Acts like a relational database
- Developed using MongoDB and mongoose
  - Could have been done as a relational database
  - Would have been easier as a relational database



USING MONGODB: USING IT LIKE A RELATIONAL DATABASE

- After setting connection
- Setup mongoose models (optional)

```
let Address = require('../models/address.model');
let Condition = require('../models/condition.model');
let DateTime = require('../models/dateTimeSystem.model');
let GeoLocation = require('../models/geoLocationSystem.model');
let PathEnvironmentalProperties = require('../models/pathEnvironmentalProperties.model');
let PathPhysicalProperties = require('../models/pathPhysicalProperties.model');
let Report = require('../models/report.model');
let ReportVersion = require('../models/reportVersion.model');
let ReputationHistory = require('../models/reputationHistory.model');
let User = require('../models/user.model');
let UserChanges = require('../models/userChanges.model');
let UserReputation = require('../models/userReputation.model');
let WeatherSystem = require('../models/weatherSystem.model');
let apiTemperature = -200.0;
let testVariable = "not initialized";
```

USING MONGODB: RETRIEVE ALL DATA

- Search for the documents

```
/**Create the response array and template for one response */
var responseArray = [];


/**Get all reports and other data and their IDs */
const reports = await Report.find(function(err,arr) {});
const addresses = await Address.find(function (err,arr){});
const dateTimes = await DateTime.find(function (err,arr){});

//const user = await User.findOne({_id:report.userId}, function(err,arr){});
const weatherSystems = await WeatherSystem.find(function (err,arr){});
const reportVersions = await ReportVersion.find(function (err,arr){});
const geoLocations = await GeoLocation.find(function (err,arr){});
const conditions  = await Condition.find(function (err,arr){});
const pathEnvironments = await PathEnvironmentalProperties.find(function (err,arr){});
const pathPhysicals = await PathPhysicalProperties.find(function (err,arr){});
```

USING MONGODB: CONTINUE

- Collect all object ids to combine data into a tabular form in a later step

```
reports.forEach(function(item, index){
    var roadReport = item;
    oneReport = {
        reportId : "",
        dateTimeId : "",
        addressId : "",
        conditionId: "",
        weatherId: "",
        reportVersionId: "",
        date : "",
        time : "",
        street : "",
        city : "",
        delay: "",
        typeMaterial : "",
        physProperties : "",
        envProperties : "",
        geoPrecision: "",
        curVersion: "",
        reputationScore: "",
        trustworthinessOfReport: "",
        additonalInfo: "",
        temperature: "",
        season: "",
        raining: "",
        photo:""
    };
    /** After finding each report, build the response array and set the important IDs */
    oneReport.reportId = roadReport._id;
    oneReport.dateTimeId = roadReport.dateTimeId;
    oneReport.addressId = roadReport.addressId;
    oneReport.conditionId = roadReport.conditionId;
    oneReport.weatherId = roadReport.weatherId;
    oneReport.reportVersionId = roadReport.reportVersionId;
    oneReport.trustworthinessOfReport = roadReport.reputationScore;
    oneReport.reputationScore = roadReport.reputationScore;
    responseArray.push(oneReport);
});
```

USING MONGODB: CONTINUE

- Search for specific documents (objects) based on the object id

```
responseArray.forEach(function(report){
    /**
     * Based on the important IDs, find all relevant information
     * from the other arrays and build the response
     */
    dateTimes.forEach(function(date) {
        if(new String(report.dateTimeId).valueOf() == new String(date._id).valueOf()) {
            let d = date.day;
            let m = date.month;
            if (parseInt(date.day) < 10) {
                d = "0" + date.day;
            }
            if (parseInt(date.month) < 10) {
                m = "0" + date.month;
            }
            report.date = d + "." + m + "." + date.year;
            report.time = date.time;
            report.delay = date.delay;
        }
    });
    reportVersions.forEach(function(version) {
        if(new String(report.reportVersionId).valueOf() == new String(version._id).valueOf()) {
            report.curVersion = version.curVersion;
        }
    });
    weatherSystems.forEach(function(weather) {
        if(new String(report.weatherId).valueOf() == new String(weather._id).valueOf()) {
            report.temperature = parseFloat(weather.temperature);
            report.season = weather.season;
            report.raining = weather.raining;
        }
    });
    addresses.forEach(function(address) {
        if(new String(report.addressId).valueOf() == new String(address._id).valueOf()) {
            report.city = address.city;
            report.street = address.street;
            geoLocations.forEach(function(geoLocation) {
                if(new String(address.geoLocationId).valueOf() == new String(geoLocation._id).valueOf()) {
                    report.geoPrecision = geoLocation.geoPrecision;
                }
            });
        }
    });
```

USING MONGODB: CONTINUE

- Finally send the data to the client

```
res.json(responseArray);
```

USING MONGODB: ADDING DATA

- Create new objects based on the models

```
let address             = new Address();

let condition           = new Condition();
let pathEnvironment     = new PathEnvironmentalProperties();
let pathPhysical        = new PathPhysicalProperties();

let geoLocation         = new GeoLocation();
let dateTime            = new DateTime();
let weather             = new WeatherSystem();

let report              = new Report();
let reportVersion       = new ReportVersion();
let userChanges         = new UserChanges();
```

USING MONGODB: ADDING DATA

- Add the new data to the objects

```
dateTime.day = today.getDate();
dateTime.month = today.getMonth()+1; //0-11 months
dateTime.year = today.getFullYear();
let hours = today.getHours() + 2; //GMT +2 offset
let minutes = today.getMinutes();
let seconds = today.getSeconds();
if (today.getHours() < 10) {
    hours = "0" + (today.getHours() + 2);
}
if (today.getMinutes() < 10) {
    minutes = "0" + today.getMinutes();
}
if (today.getSeconds() < 10) {
    seconds = "0" + today.getSeconds();
}
dateTime.time = hours + ":" + minutes + ":" + seconds;

address.street = data.street;
address.city = data.city;
address.geoLocationId = geoLocation._id;
geoLocation.latitude = data.latitude;
geoLocation.longitude = data.longitude;
geoLocation.geoPrecision = data.geoPrecision;

weather.raining = "No rain";
weather.season = "Spring";
weather.temperature = apiTemperature;

condition.pathPhysicalPropertiesId = pathPhysical._id;
condition.pathEnvironmentalPropertiesId = pathEnvironment._id;
condition.additionalInfo = data.additionalInfo;
```

USING MONGODB: ADDING DATA

- Check if something exists (and act accordingly)

```
let user = null;
let existingUser = null;
try {
    existingUser = await User.findOne({userName: data.userName}, function (err, res){});
} catch (error) {
    existingUser = null;
}

if (existingUser == null) {
    user = new User();
} else {
    user = existingUser;
}
```

USING MONGODB: ADDING DATA

- Save the objects to the database

```
address.save(function(err) {if (err) console.log(err); });
geoLocation.save(function(err) {if (err) console.log(err); });
dateTime.save(function(err) {if (err) console.log(err); });
weather.save(function(err) { if (err) console.log(err); });
report.save(function(err) { if (err) console.log(err); });
condition.save(function(err) { if (err) console.log(err); });
pathPhysical.save(function(err) { if (err) console.log(err); });
pathEnvironment.save(function(err) { if (err) console.log(err); });
userChanges.save(function(err) {if (err) console.log(err); });
reportVersion.save(function(err) { if (err) console.log(err); });
```

USING MONGODB: EDITING DATA

- Find the objects you want to edit

```
/**First find the one information based on the given ID */
let report = await Report.findOne({_id:id}, function(err,arr){});
let user = await User.findOne({userName:data.userName}, function(err,arr){});
let address = await Address.findOne({_id:report.addressId}, function(err,arr){});
let condition  = await Condition.findOne({_id:report.conditionId}, function(err,arr){});
let reportVersion = await ReportVersion.findOne({_id:report.reportVersionId}, function(err,arr){});

let pathEnvironment = await PathEnvironmentalProperties.findOne({_id:condition.pathEnvironmentalPropertiesId}, function(err,arr){});
let pathPhysical = await PathPhysicalProperties.findOne({_id:condition.pathPhysicalPropertiesId}, function(err,arr){});
```

USING MONGODB: EDITING DATA

- Update the given data

```
address.street = data.street;
address.city = data.city;
condition.additionalInfo = data.additionalInfo;

pathPhysical.pathType = data.pathType;
pathPhysical.pathMaterial = data.pathMaterial;
pathPhysical.cracks = data.cracks;
pathPhysical.holes = data.holes;
pathPhysical.debree = data.debree;

pathEnvironment.ice = data.ice;
pathEnvironment.snow = data.snow;
pathEnvironment.sand = data.sand;
pathEnvironment.puddles = data.puddles;
pathEnvironment.leaves = data.leaves;
pathEnvironment.pollen = data.pollen;
```

USING MONGODB: EDITING DATA

- Save the new data

```
/**
 * All the save functions
 */
address.save(function(err) {if (err) res.send(err); });
report.save(function(err) { if (err) res.send(err); });
condition.save(function(err) { if (err) res.send(err); });
pathPhysical.save(function(err) { if (err) res.send(err); });
pathEnvironment.save(function(err) { if (err) res.send(err); });
reportVersion.save(function(err) { if (err) res.send(err); });
```

FINAL NOTES

- Using a NoSQL database in the same fashion as a relational database has its downsides
  - Throwing away the benefits of a NoSQL and doing an inferior version of a relational database
- The major benefit of a NoSQL database:

1. Asynchronous operations
   - User can continue using the interface while database retrieves or sends data
   - In the case of failure, manual checks and rollbacks have to be done
2. Modifying the models on during development


LUT
University