(Not So) Far Beyond NF²

Data Model for Structured Documents

Guillaume Raschia — Nantes Université

Last update: October 17, 2023

Contents

Relaxing NF^2

Doc Encoding

Doc Modeling

[Source : S. Abiteboul, SIGMOD/PODS Anniversary 2011]

[Source : P. Rigaux, `b3d.bdpedia.fr`]

From NF^2 to Documents

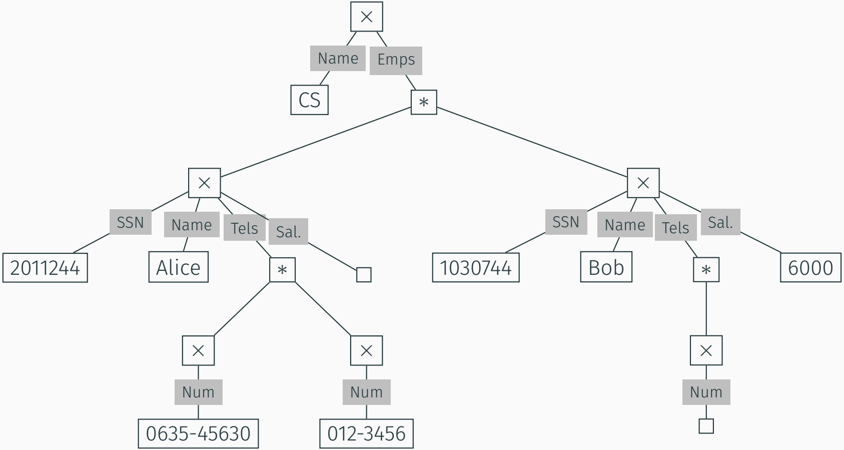
Complex Object Model

The Departments table

<table>
<tr><th>Name</th><th>Employees</th></tr>
<tr><td>Computer Science</td><td>
<table>
<tr><th>SSN</th><th>Name</th><th>Telephones</th><th>Salary</th></tr>
<tr><td>20011244</td><td>Alice</td><td>
<table>
<tr><th>Num</th></tr>
<tr><td>0635-45630</td></tr>
<tr><td>012-3456</td></tr>
</table>
</td><td>NULL</td></tr>
<tr><td>1030744</td><td>Bob</td><td>
<table>
<tr><th>Num</th></tr>
<tr><td>NULL</td></tr>
</table>
</td><td>6,000</td></tr>
</table>
</td></tr>
</table>

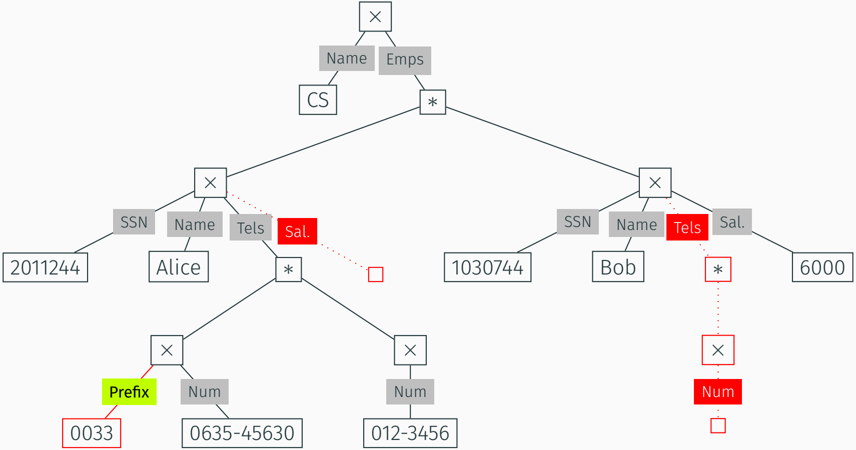
Complex Object Model (cont'd)

Tuple and Set type constructors are used freely



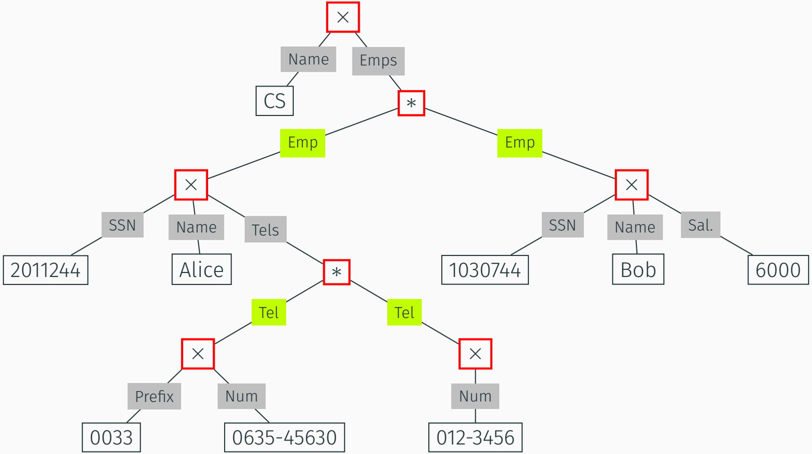
First Revolution

More flexibility: **schema-less** approach



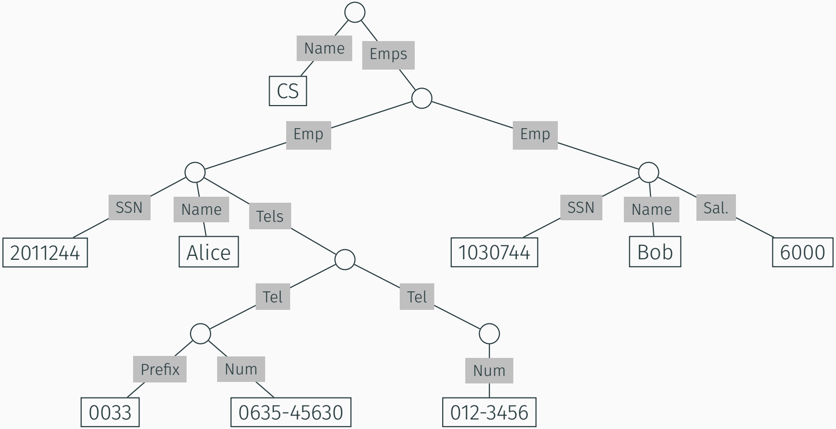
Second Revolution

Remove intermediate node values and label all edges



Semistructured Data aka. Structured Documents

Labeled Trees



Properties of the Labeled Trees

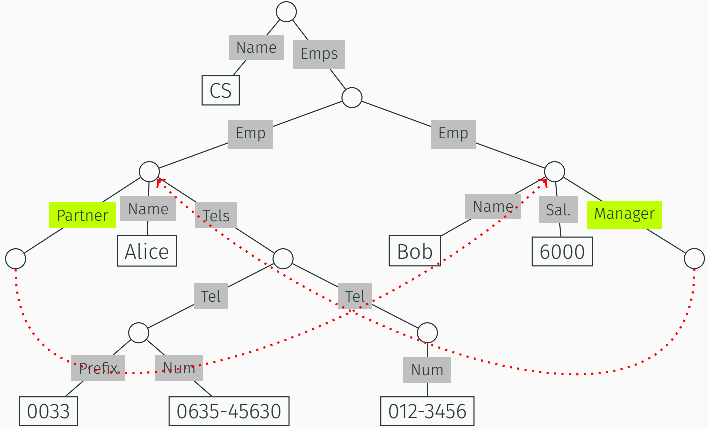
- **Unranked**: like nested relations
- **Unbounded**: unlike nested relations
- **Ordered**: inherited from the document community

From the database perspective, order is painful for optimization

- Beyond the scene: Tree Automata Theory

When Trees Become Graphs!

Introducing **references**: cycles and many other issues...



Structured Documents: What for?

Semistructured data are well-suited for the web:

- data exchange over the http protocol
- web services and API implementation
- both human and machine readable
- low-level—programming language dependant—interface

The Best Time to Plant a Tree Was 20 Years Ago. The Second Best Time is now.

Chinese Proverb

Encoding

Requirements

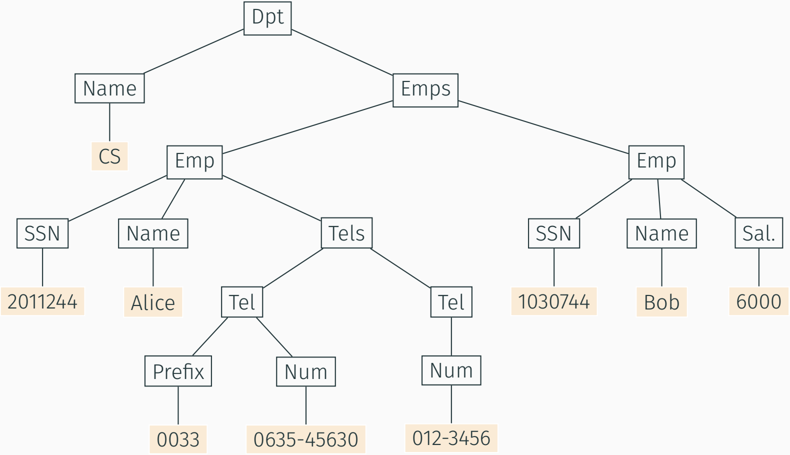
Structured documents are:

- **Self-described**: schema embedded into the document/data itself
- **Complex**: built with nested records and sets
- **Schema-less**: neither pre-defined structure nor mandatory typing
- **Serializable** into a self-contained string

Popular Languages

- eXtensible Markup Language
- JavaScript Object Notation (and Binary JSON)
- YAML Ain't Markup Language
- Protobuf

XML Encoding



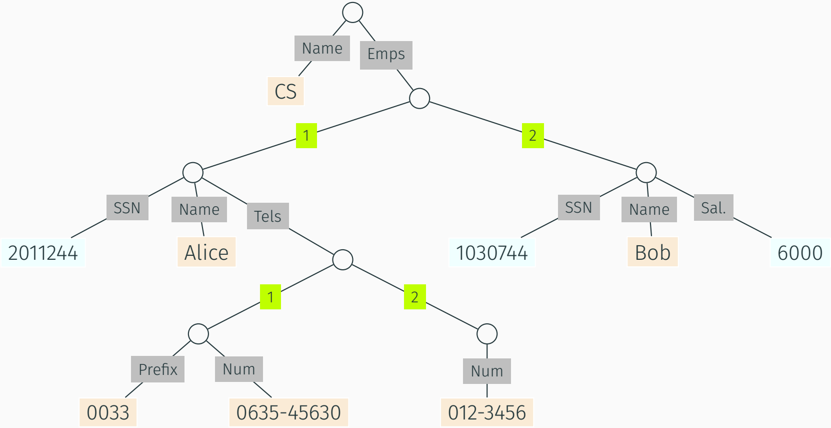
Diff w.r.t. the conceptual labeled tree

1. Root node - 2. Node labels - 3. Node types (Element or Text)

XML Encoding (cont’d)

```
<DEPT>
    <NAME>CS</NAME>
    <EMPLOYEES>
        <EMP>
            <SSN>2011244</SSN>
            <NAME>Alice</NAME>
            <TELS>
                <TEL><PREFIX>0033</PREFIX><NUM>0635-45630</NUM><TEL>
                <TEL>012-3456</TEL>
            </TELS>
        </EMP>
        <EMP>
            <SSN>1030744</SSN>
            <NAME>Bob</NAME>
            <SALARY>6,000</SALARY>
        </EMP>
    </EMPLOYEES>
</DEPT>
```

JSON Encoding



Diff w.r.t. the conceptual labeled tree

1. No two identical keys - 2. Arrays - 3. Leaf node types

JSON Encoding (cont’d)

```
{
    "name": "CS",
    "employees": [
        {
            "ssn": 2011244,
            "name": "Alice",
            "tels": [ { "prefix": "0033",
                        "num": "0635-45630" },
                      { "num": "012-3456" }
                    ]
        },
        {
            "ssn": 1030744,
            "name": "Bob",
            "salary": 6000
        },
    ]
}
```

YAML Encoding (for fun)

```yaml
---
name: CS
employees:
    - ssn:    2011244
      name:   Alice
      tels:
        - prefix: 0033
          num:    0635-45630
        - num:    012-3456
    - ssn:    1030744
      name:   Bob
      salary: 6000
...
```

1. Similar to JSON - 2. Introduce References &/* - 3. With many other nice features

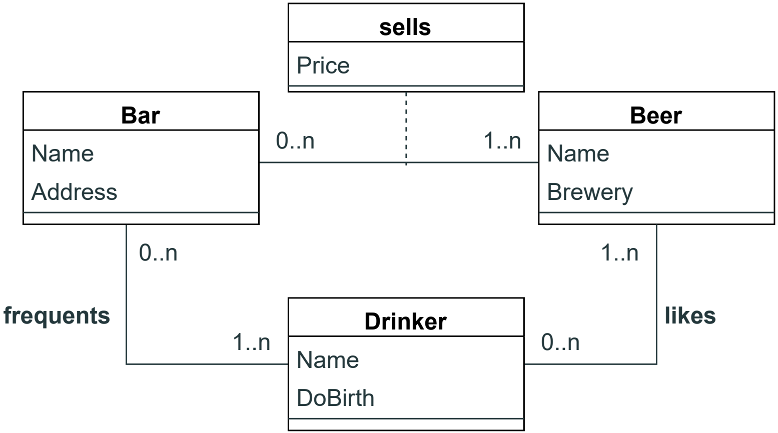
Modeling

Requirements for Structured Document Modeling

- Docs are mainly **nested key-value pairs**
- Docs come with a **–surrogate– key** such like “`_id`”
- Docs are stored into **Collections**

Bars & Beers & Drinkers

The very famous Stanford TCB Example



Relational Model of B-B-D

```
Bars(name: "Live Bar", address: "6 rue de Strasbourg, 44000 Nantes")
Beers(name: "Trompe Souris", brewery: "La Divatte")
Beers(name: "Titan", brewery: "Bouffay")
Drinkers(name: "Alice", dob: 2001-09-10)
Drinkers(name: "Bob", dob: 1998-04-23)

Sells(bar: "Live Bar", beer: "Trompe Souris", price: 3.0)
Sells(bar: "Live Bar", beer: "Titan", price: 2.5)

Likes(drinker: "Alice", beer: "Titan")
Likes(drinker: "Bob", beer: "Trompe Souris")
Likes(drinker: "Bob", beer: "Titan")

Frequents(drinker: "Alice", bar: "Live Bar")
Frequents(drinker: "Bob", bar: "Live Bar")
```

Primary and Foreign Keys—aka. integrity constraints—everywhere

Aggregate Model of B-B-D

```
{ "_id": "Live Bar",          // key with unique index
  "address": "6 rue de Strasbourg, 44000 Nantes",
  // array of drinkers (embedded docs) that frequent the Live Bar
  "drinkers_frequent": [
      { "drinker_id": "Alice",
        "dob":  "2001-09-10",
        // array of beers liked (embedded docs) by Alice
        "likes": [
            {"beer_id": "Titan", "brewery": "Bouffay"} ] },
      { "drinker_id": "Bob",
        "dob":   "1998-04-23",
        "likes": [
            { "beer_id": "Trompe Souris", "brewery": "La Divatte"},
            { "beer_id": "Titan", "brewery": "Bouffay" } ] } ],
  // array of beers (embedded docs) sold in the Live Bar
  "beers_sold": [
      { "beer_id": "Trompe Souris",
        "brewery": "La Divatte",
        "price": 3.0 },
      { "beer_id": "Titan",
        "brewery": "Bouffay",
        "price": 2.5 } ] }
```

The Aggregate Model

- Substitute foreign keys by **nested documents**
- Embed in doc each and every parts of the data unit

Pros

- **No more joins**
- Autonomous data unit designed to be **distributed** across shards
- **No more transactions**: atomic reads and writes for single docs

Welcome to the NoSQL World!

The Aggregate Model (cont'd)

We encoded the **Bar**'s view point...

Let's give a try to the **Drinker**'s one

```
{ "_id": "Alice",                     // unique key
  "dob": "2001-09-10",
  // array of bars (embedded docs) that Alice frequents
  "frequents": [
      { "bar_id": "Live Bar",        // bar's key: no referential integrity check
        "address": "3 rue de Strasbourg, 44000 Nantes",
        // array of beers (embedded docs) sold by the Live Bar
        "beers_sold": [
          { "beer_id": "Trompe Souris",
            "brewery": "La Divatte",
            "price": 3.0 },
          { "beer_id": "Titan",                    // first occurrence of Titan beer
            "brewery": "Bouffay",
            "price": 2.5 } ] } ],
  // array of beers (embedded docs) that Alice likes
  "likes": [
      { "beer_id": "Titan",                        // second occurrence of Titan beer
        "brewery":  "Bouffay" } ] }
```

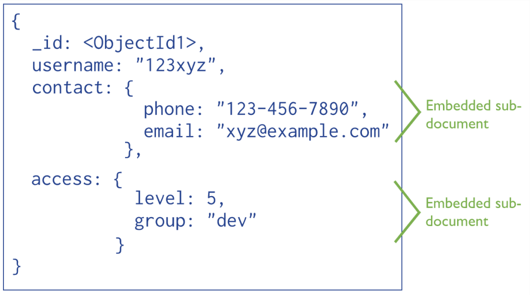
No Free Lunch

What about

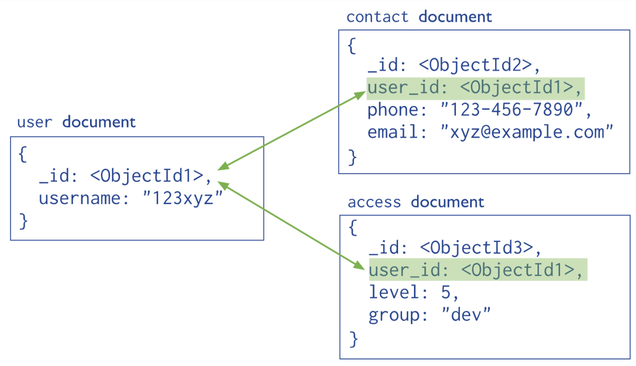
- Retrieving all the breweries?
  → scan the complete collection, remove duplicates!
- Removing Trompe Souris from the Live Bar?
  → loose La Divatte brewery info and
  → may create dangling references somewhere
- Updating Titan to Moustache ?!
  → probable inconsistent duplicates elsewhere

MongoDB Data Model Design

Embedded Data Model vs. Normalized Data Model



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



user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Source: official MongoDB documentation

Remind that there is **no foreign key** in the Normalized Data Model

The Right Trade-Off

Design of Relationships

- One-to-One: embedded
- One-to-Many: mainly embedded, but it depends...
- Many-to-Many: it depends...

How far one needs to denormalize for performance reason ?!

Same question arises in RM Design

No Free Lunch (cont’d)

About Schema-less Modeling

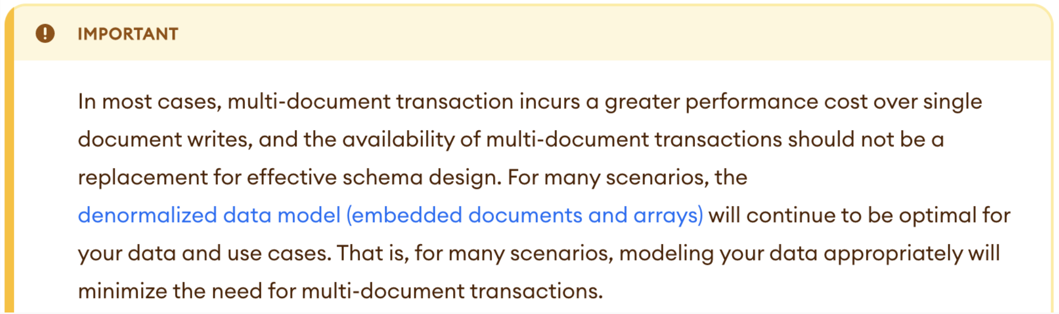
From the official MongoDB documentation

> This flexibility facilitates the mapping of documents to an entity or an object. Each document can match the data fields of the represented entity, even if the document has substantial variation from other documents in the collection.
>
> In practice, however, the documents in a collection share a similar structure, and you can enforce document validation rules for a collection during update and insert operations. See Schema Validation for details.

MongoDB supports **JSON Schema** validation, on a *Collection* basis

MongoDB Data Model Design

Atomic –single document– Tx vs. Multi-document Tx



❗ **IMPORTANT**

In most cases, multi-document transaction incurs a greater performance cost over single document writes, and the availability of multi-document transactions should not be a replacement for effective schema design. For many scenarios, the denormalized data model (embedded documents and arrays) will continue to be optimal for your data and use cases. That is, for many scenarios, modeling your data appropriately will minimize the need for multi-document transactions.

Source: MongoDB official documentation

Query Language in MongoDB

From the MongoDB official documentation

```
db.people.aggregate(
        [ { $group : {
                 _id : "$status"
         } } ]
)
```

```
SELECT DISTINCT(status)
FROM people
```

```
db.users.aggregate([{$lookup:
    {
     from: "products",
     localField: "product_id",
     foreignField: "_id",
     as: "products"
    }
}])
```

```
SELECT *
FROM users
LEFT JOIN products
ON users.product_id = products._id
```

MongoDB Join

```
SELECT o.*, w.warehouse, w.instock FROM warehouses w
JOIN orders o ON w.stock_item = o.item AND w.instock >= o.ordered
```

```
db.orders.aggregate( [ { $lookup:
                {
            from: "warehouses",
            let: { order_item: "$item", order_qty: "$ordered" },
            pipeline: [
                           { $match: { $expr: { $and:
                        [
                                { $eq: [ "$stock_item",  "$$order_item" ] },
                                 { $gte: [ "$instock", "$$order_qty" ] }
                               ]
                }}},
                      { $project: { stock_item: 0, _id: 0 } }
           ],
           as: "stockdata"
        }
} ] )
```

OQL (Obfuscated Query Language) should be the name...

No Free Lunch: Main Points

In the JSON World[1]:

- Design driven by the **data access model**: how the app consume the data
- Lots of **redundancies** into/between aggregates and collections
- Lots of—possible—**anomalies**
- No(t Yet a) Declarative Query Language: complicated and adhoc statements
- **No referential integrity checking**: to do in app
- Essentially **no type—schema—checking**: to do in app
- **No Tx** also yields to **inconsistencies**

NoSQL is a DIY World!

[1] XML galaxy is far better, even if it shares the design dilemma in the first place.

JSON vs. XML

XML Maturity

- **W3C open standard** with a very large spec
- formal data model: **Document Object Model** (DOM)
- declarative FLWR-based language: **XQuery/XPath**
- Schema languages: **DTD**, XML Schema, RelaxNG

XML Technology supports large docs and node-based processing

- see in action: JS React mount/unmount components (DOM subtrees)

Native XML Databases

Representatives

MarkLogic™
OPENLINK VIRTUOSO UNIVERSAL SERVER
sedna
Native XML Database System
Tamino XML Server
existdb

JSON vs. XML (cont'd)

JSON Popularity

- 5 pages long ECMA Standard: `https://www.json.org/`
- lightweight data-interchange format: CRUD with Web API (REST or GraphQL)
- easy to parse in any PL: nested dicts and arrays
- on its way to:
  - a formal **data model**

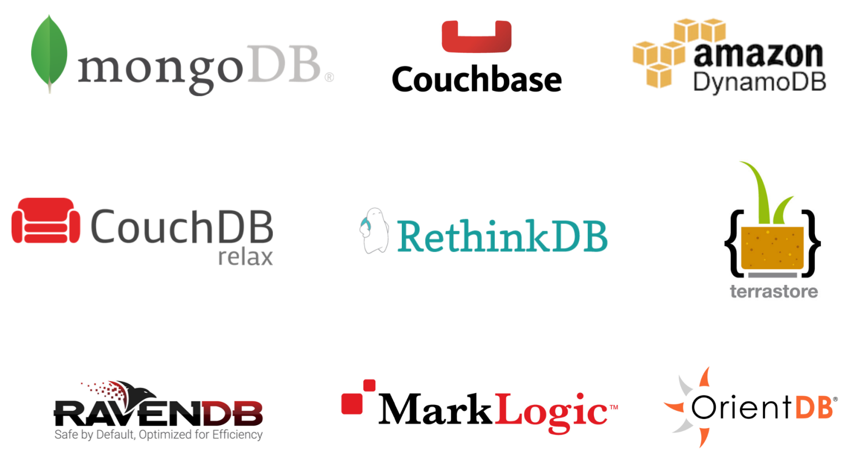
    P. Bourhis et al. (2017) *JSON: Data model, Query languages and Schema specification.* PODS 2017.
  - **JSON Schema**: `http://json-schema.org/`
  - FLWR-based **query languages**: SQL++, JSONiq (and JSONPath)

JSON Document stores aim at handling collections of small docs

No "pure JSON" Store but BSON Store instead

Document Stores

Representatives


mongoDB®
Couchbase
amazon DynamoDB
CouchDB relax
RethinkDB
terrastore
RAVENDB Safe by Default, Optimized for Efficiency
MarkLogic™
OrientDB®