Document Databases

DS3000: Foundations of Data Science

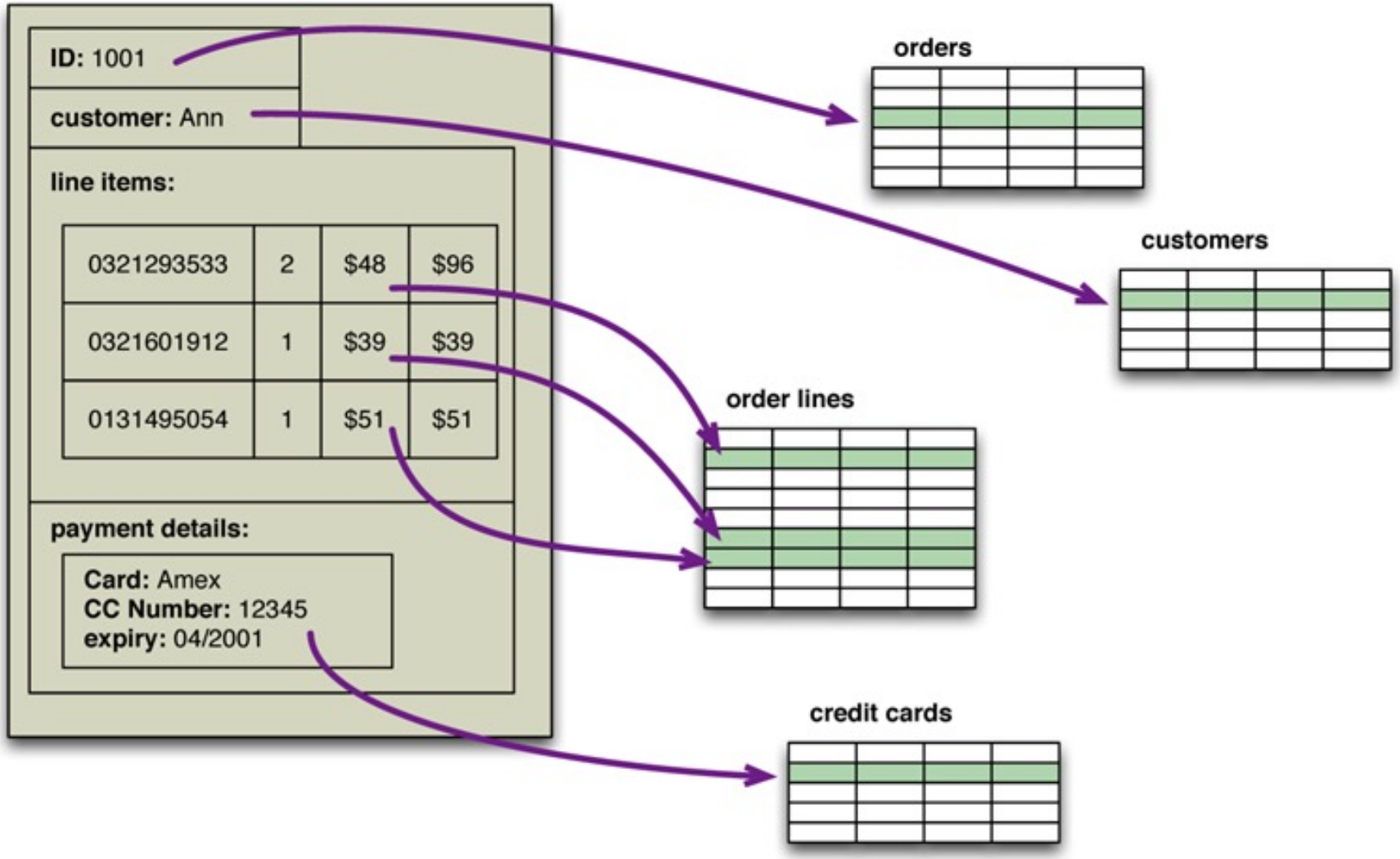
DS5110: Data Management and Processing

db-engines.com

364 systems in ranking, February 2021

Rank			DBMS	Database Model	Score		
Feb 2021	Jan 2021	Feb 2020			Feb 2021	Jan 2021	Feb 2020
1.	1.	1.	Oracle	Relational, Multi-model	1316.67	-6.26	-28.08
2.	2.	2.	MySQL	Relational, Multi-model	1243.37	-8.69	-24.28
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1022.93	-8.30	-70.81
4.	4.	4.	PostgreSQL	Relational, Multi-model	550.96	-1.27	+44.02
5.	5.	5.	MongoDB	Document, Multi-model	458.95	+1.73	+25.62
6.	6.	6.	IBM Db2	Relational, Multi-model	157.61	+0.44	-7.94
7.	7.	↑ 8.	Redis	Key-value, Multi-model	152.57	-2.44	+1.15
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	151.00	-0.25	-1.16
9.	9.	↑ 10.	SQLite	Relational	123.17	+1.28	-0.19
10.	10.	↑ 11.	Cassandra	Wide column	114.62	-3.46	-5.74

Impedance mismatch problem



Object vs. Relational Databases

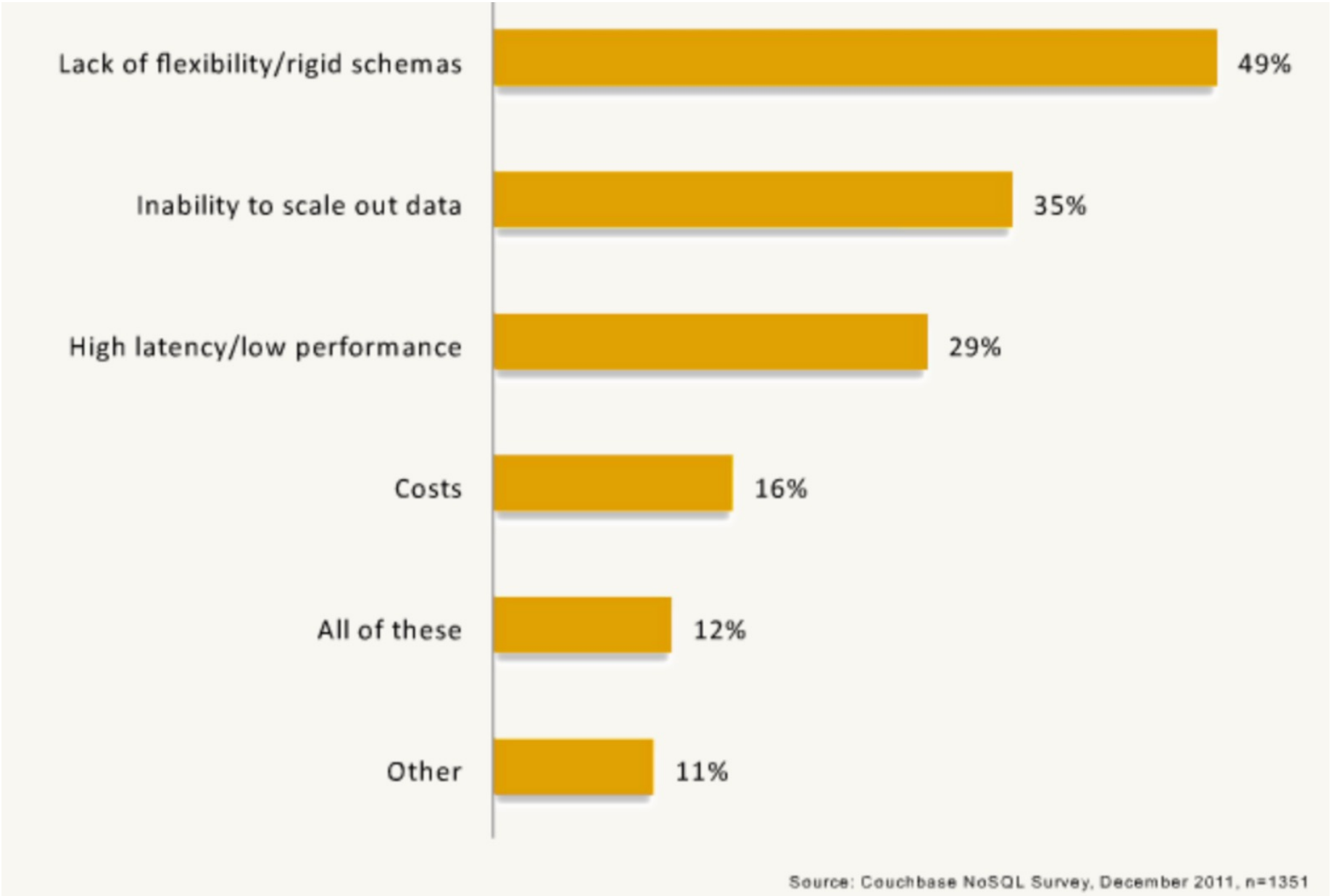
A relational database is like a garage that forces you to take your care apart and store the pieces in little drawers.

-- Object-oriented data community, mid-1990s

An object database is like a closet which requires that you hang up your suit, with tie, underwear, belt, socks and shoes all attached.

-- David Ensor, same period.

Drivers of NoSQL



Relational DB vs. Document DB

RDBMS		MongoDB
Database	➡	Database
Table, View	➡	Collection
Row	➡	Document (BSON)
Column	➡	Field
Index	➡	Index
Join	➡	Embedded Document
Foreign Key	➡	Reference
Partition	➡	Shard

- A collection is not strict about what it stores (Schema-less)
- Hierarchy is evident in the design
- Embedded Document to represent relation.

Document Databases

A **document database** is a non-relational database that stores data as structured documents, usually XML or JSON formats.

Designed to be *simple*, *flexible*, and *scalable*

```
{ "users":[
        {
            "firstName":"Ray",
            "lastName":"Villalobos",
            "joined": {
                "month":"January",
                "day":12,
                "year":2012
            }
        },
        {
            "firstName":"John",
            "lastName":"Jones",
            "joined": {
                "month":"April",
                "day":28,
                "year":2010
            }
        }
    ]}
```

BSON (Binary JSON)

BSON *[bee · sahn]*, short for Binary JSON, is a binary-encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.

BSON can be compared to binary interchange formats, like Protocol Buffers. BSON is more "schema-less" than Protocol Buffers, which can give it an advantage in flexibility but also a slight disadvantage in space efficiency (BSON has overhead for field names within the serialized data).

BSON was designed to have the following three characteristics:

1. **Lightweight**

   Keeping spatial overhead to a minimum is important for any data representation format, especially when used over the network.

2. **Traversable**

   BSON is designed to be traversed easily. This is a vital property in its role as the primary data representation for MongoDB.

3. **Efficient**

   Encoding data to BSON and decoding from BSON can be performed very quickly in most languages due to the use of C data types.

JSON

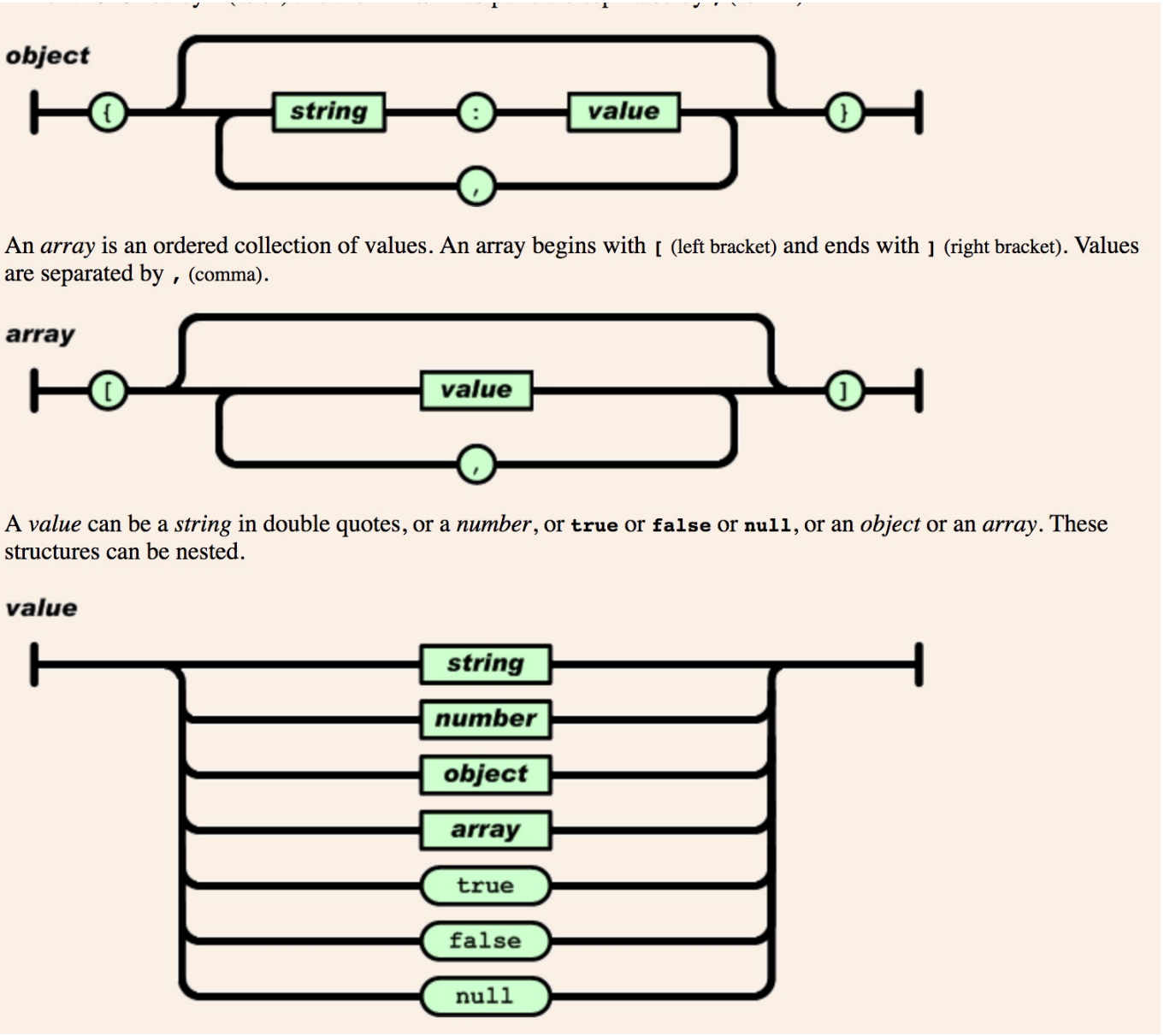
JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

JSON is built on two structures:

•A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.

•An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.



BSON Specification 1.1

Specification Version 1.1

BSON is a binary format in which zero or more ordered key/value pairs are stored as a single entity. We call this entity a *document*.

The following grammar specifies version 1.1 of the BSON standard. We've written the grammar using a pseudo-BNF syntax. Valid BSON data is represented by the `document` non-terminal.

Basic Types

The following basic types are used as terminals in the rest of the grammar. Each type must be serialized in little-endian format.

`byte`	1 byte (8-bits)
`int32`	4 bytes (32-bit signed integer, two's complement)
`int64`	8 bytes (64-bit signed integer, two's complement)
`uint64`	8 bytes (64-bit unsigned integer)
`double`	8 bytes (64-bit IEEE 754-2008 binary floating point)
`decimal128`	16 bytes (128-bit IEEE 754-2008 decimal floating point)

Non-terminals

The following specifies the rest of the BSON grammar. Note that quoted strings represent terminals, and should be interpreted with C semantics (e.g. `"\x01"` represents the byte `0000 0001`). Also note that we use the `*` operator as shorthand for repetition (e.g. `("\x01"*2)` is `"\x01\x01"`). When used as a unary operator, `*` means that the repetition can occur 0 or more times.

`document ::= int32 e_list "\x00"`	BSON Document. int32 is the total number of bytes comprising the document.
`e_list ::= element e_list`	

JSON → BSON

The following are some example documents (in JavaScript / Python style syntax) and their corresponding BSON representations.

```
{"hello": "world"}                  →   \x16\x00\x00\x00                    // total document size
                                        \x02                               // 0x02 = type String
                                        hello\x00                          // field name
                                        \x06\x00\x00\x00world\x00          // field value
                                        \x00                               // 0x00 = type EOO ('end of object')
```

```
{"BSON": ["awesome", 5.05, 1986]} →   \x31\x00\x00\x00
                                      \x04BSON\x00
                                      \x26\x00\x00\x00
                                      \x02\x30\x00\x08\x00\x00\x00awesome\x00
                                      \x01\x31\x00\x33\x33\x33\x33\x33\x33\x14\x40
                                      \x10\x32\x00\xc2\x07\x00\x00
                                      \x00
                                      \x00
```

XML (eXtensible Markup Language)

- Used as an information storage standard
- XML + CSS (Cascading Style Sheets) enables development of websites that separate data and format
- The foundation for web-service specifications such as SOAP
- A rich ecosystem of tools and standards that includes:

Xpath – Retrieving specific elements of an XML document

Xquery – Interrogating XML documents

XML schema – Describes XML documents within a namespace and used for document validation

XSLT – for transforming XML into other formats such as HTML

DOM (Document Object Model) – An OO API for navigating XML docs.

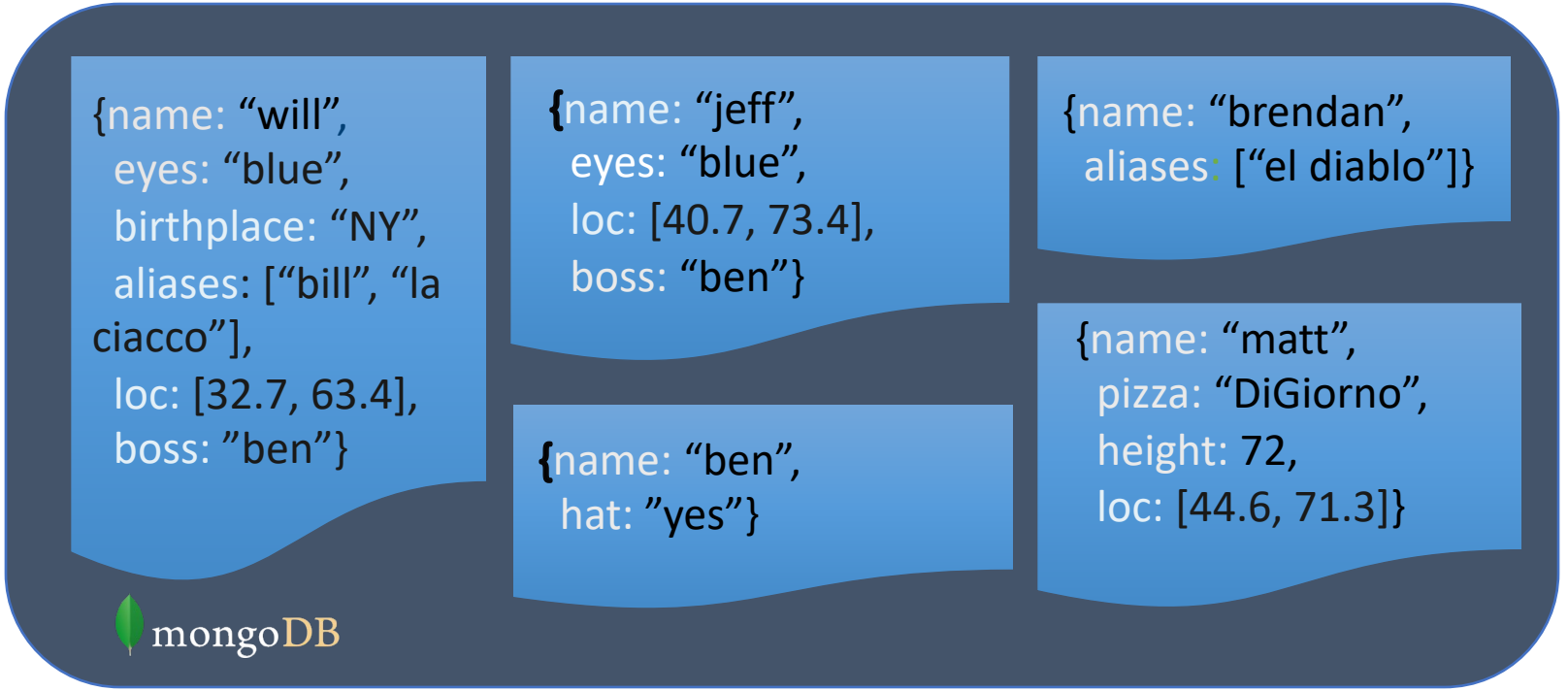
```
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

Why have document databases flourished since 2008?

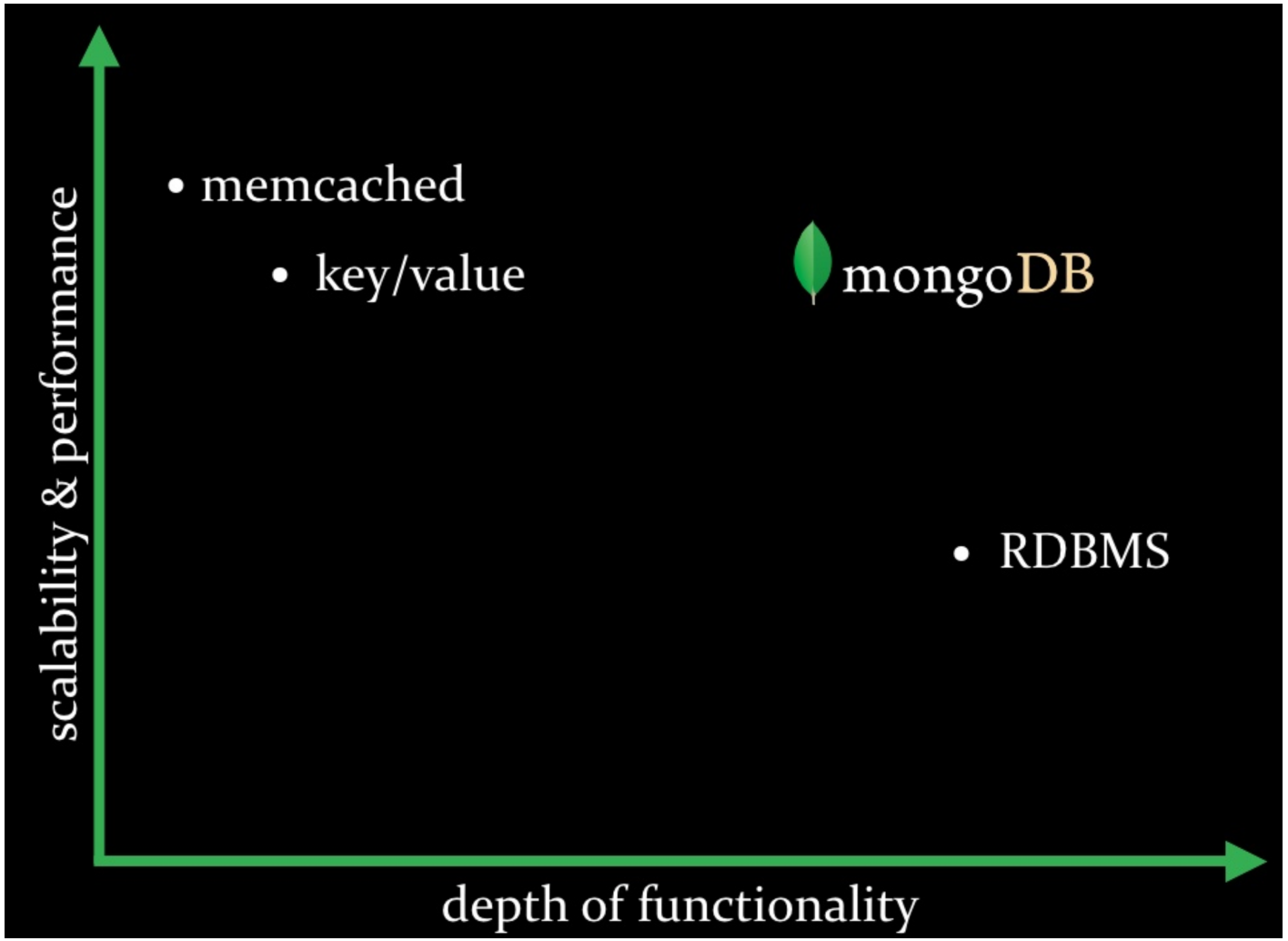
- They address the *impedance mismatch* problem associated with object-oriented developers needing to persist their data to relational databases. (Object-Relational Mapping)
- Self-describing formats support *ad hoc* query by value absent in pure key-value stores.
- Aligned with web-service based programming models using JSON / XML as a transport layer.

Schema-less

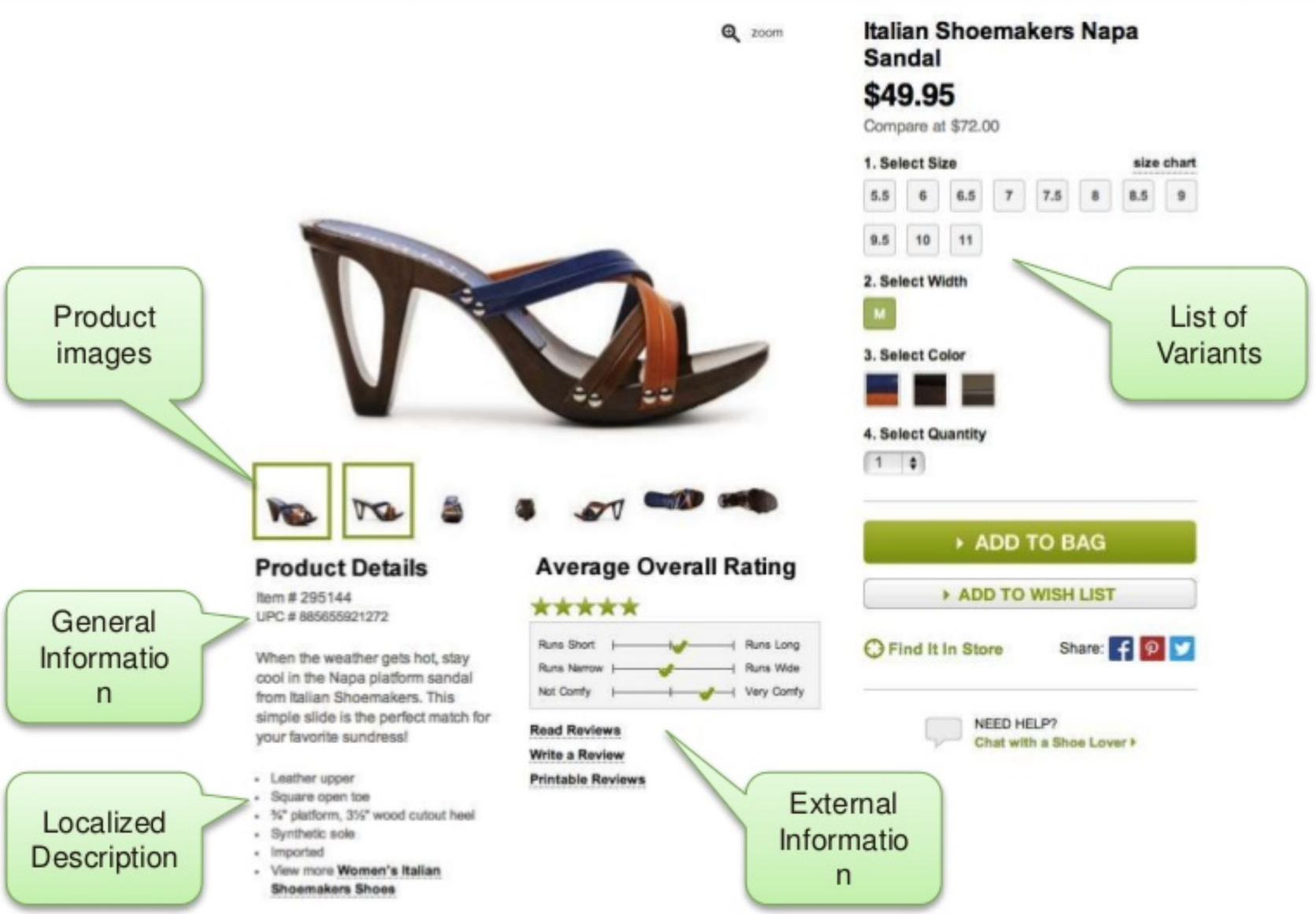
- MongoDB does not need any pre-defined data schema
- Every document in a collection could have different data



Document databases: A happy medium

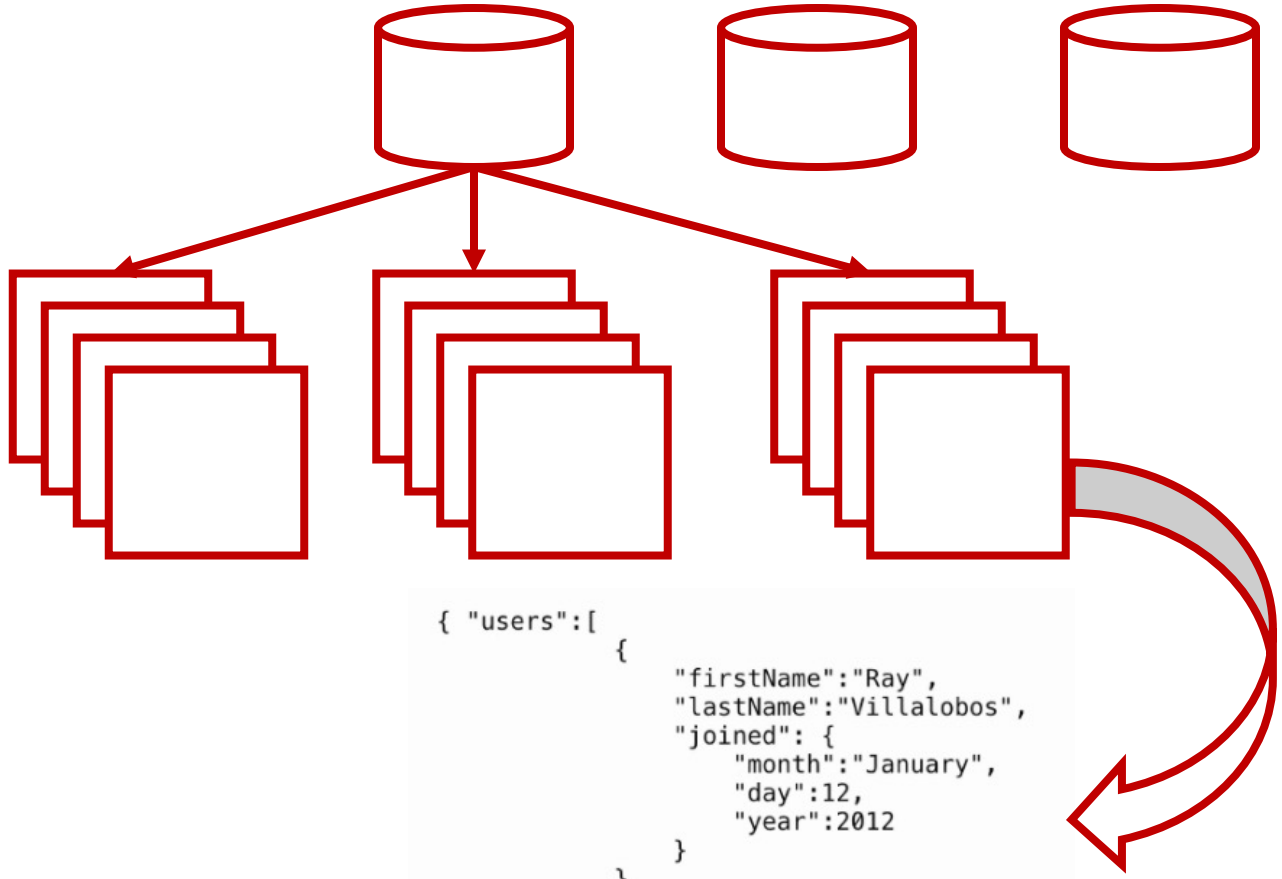


Product Catalogs / e-commerce websites



MongoDB: Hierarchical Objects

- A MongoDB instance may have zero or more ‘databases’
- A database may have zero or more ‘collections’.
- A collection may have zero or more ‘documents’.
- A document may have one or more ‘fields’.
- MongoDB ‘Indexes’ function much like their RDBMS counterparts.



```
{ "users":[
            {
                "firstName":"Ray",
                "lastName":"Villalobos",
                "joined": {
                    "month":"January",
                    "day":12,
                    "year":2012
                }
            },
            {
                "firstName":"John",
                "lastName":"Jones",
                "joined": {
                    "month":"April",
                    "day":28,
                    "year":2010
                }
            }
    ]}
```

What is MongoDB?

- Developed by MongoDB, Inc. (Formally 10gen Inc.)
  - Founded in 2007
  - IPO October 2017
- A document-oriented NoSQL database
  - Hash-based, *schema-less database*
  - No Data Definition Language
  - Uses BSON (Binary JSON) format
- Written in C++
- Supports APIs (drivers) in many computer languages: JavaScript, Python, Ruby, Perl, Java, Java Scala, C#, C++, Haskell, Erlang

MongoDB Features

- **Rich query support**: We can query the database as we do with SQL databases. It has a large query set that supports insert, update, delete and select operations. MongoDB supports fields, range queries, and regular expressions. Queries also support the projection where they return a value for specific keys.
- **Indexing**: MongoDB supports primary and secondary indices in its fields.
- **Replication**: Replication means providing more than one copy of data. MongoDB provides multiple copies of data with multiple servers. It provides fault tolerance, if one database server goes down, the application uses other database servers.

MongoDB Features

- **Load balancing**: Replica sets provide multiple copies of data. MongoDB can scale read operation by client request directly to the secondary node. This divides loads across multiple servers.
- **File storage**: We can store documents up to 6 MB directly to the MongoDB JSON field. For documents exceeding the size limit of 16 MB, MongoDB provides GridFS to store in chunks.
- **Aggregation**: The aggregate function takes a number of records and calculates single results like sum, min, and max. MongoDB provides a data pipeline and multistage pipeline to move large data to the aggregate function which improves performance.

Flexible product descriptions

```
➤ db.item.findOne()

{   _id: "301671", // main item id
    department: "Shoes",
    category: "Shoes/Women/Pumps",
    brand: "Guess",
    thumbnail: "http://cdn…/pump.jpg",
    image: "http://cdn…/pump1.jpg", // larger version of thumbnail
    title: "Evening Platform Pumps",
    description: "Those evening platform pumps put the perfect
finishing touches on your most glamourous night-on-the-town
outfit",
    shortDescription: "Evening Platform Pumps",
    style: "Designer",
    type: "Platform",
    rating: 4.5, // user rating
    lastUpdated: Date("2014/04/01"), // last update time
    … }
```

Product Search

- **Get item by id**

```
db.item.findOne( { _id: "301671" } )
```

- **Get item from Product Ids**

```
db.item.findOne( { _id: { $in: ["301671", "301672" ] } } )
```

- **Get items by department**

```
db.item.find({ department: "Shoes" })
```

- **Get items by category prefix**

```
db.item.find( { category: /^Shoes\/Women/ } )
```

- **Indices**

  productId, department, category, lastUpdated

Compound Libraries

- Store Chemical Compounds – Fingerprints
- Want to find compounds which are “close” to a given compound
- Need to return quickly a small set of reasonable candidates
- Few researchers working concurrently
- Use Tanimoto association coefficient to compare two compounds based on their common fingerprints


```
{
  "compound_cid" : "46200001" ,
  "smiles" :
  "CCC1C(C(C(C(=NOCC=CCN2CCCCC2)C(CC(C(C(C(C(C(=O)O1)C)OC3CC(C(C(O3)C)O)
    (C)OC)C)OC4C(C(CC(O4)C)N(C)CC5=CC=CC=C5)O)(C)O)C)C)O)(C)O" ,
  "fingerprint_count" : 120 ,
  "fingerprints" : [
    "0[N]1[C O]2[C C C]" ,
    "0[N]1[C O]2[C C C]3[C C C C C]" ,
    "0[C]1[C C C]2[C C N O]3[C C C C O O]" ,
    "0[C]1[C C C]2[C C C C O]3[C C N O]" ,
    "0[O]1[C]2[C O]3[C C C]" ,
    "0[C]1[C O O]2[C C C O]" ,
    "0[C]1[C C]2[C C]" ,
    "0[C]1[C]2[C]3[C O]" ,
    "0[C]1[C C N]2[C C C C O]3[C C C O]" ,
    ... ] ,
}
```


$$T = \frac{N_{AB}}{N_A + N_B - N_{AB}}$$

Genomics

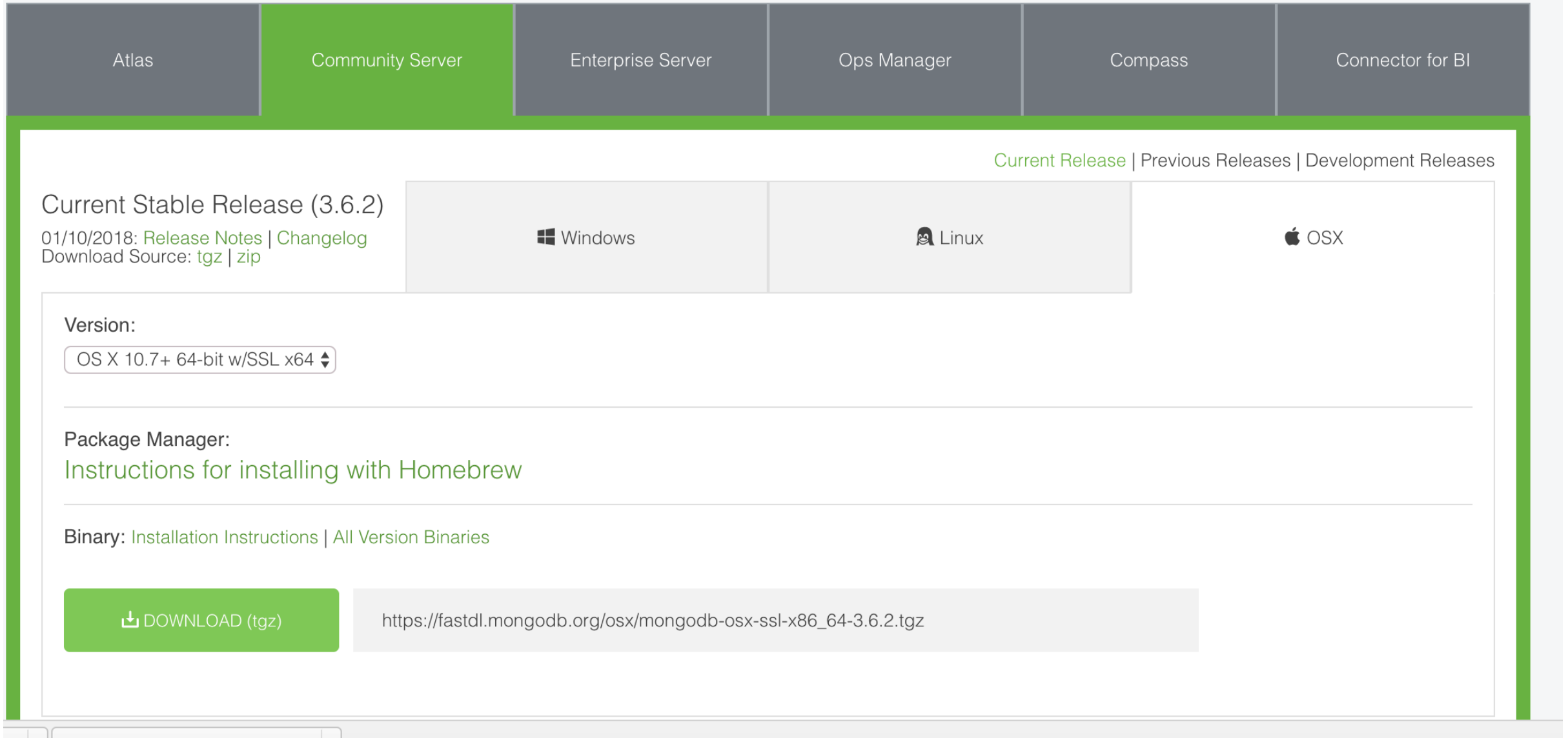
- Very large base of DNA sample sequences
  - Origin, collection method, sequence, date, …
- Enumeration of mutations relative to reference sequence
  - Positions, mutation type, base
- Need to retrieve efficiently all sequences showing a particular mutation

	1	2	3	4	5	6		7	9	10
reference	T	T	C	C	T	A	-	T	T	C
sample	T	G	C	C	-	A	G	T	T	T

- Similar to Content Management System pattern
- Add tag array in sequence document with mutation names
- Index tag array
- Queries looking for affected sequences are indexed and very fast
- Easy to setup, flexible representation and details for sequences, flexible evolution
- Can scale to massive volumes

mongoDB

MongoDB Installation



Or on Mac OSX: $ brew install mongodb

MongoDB Detailed Instructions

https://docs.mongodb.com/manual/administration/install-community/

Install MongoDB > Install MongoDB Community Edition

Install MongoDB Community Edition

These documents provide instructions to install MongoDB Community Edition.

Install on Linux

Install MongoDB Community Edition and required dependencies on Linux.

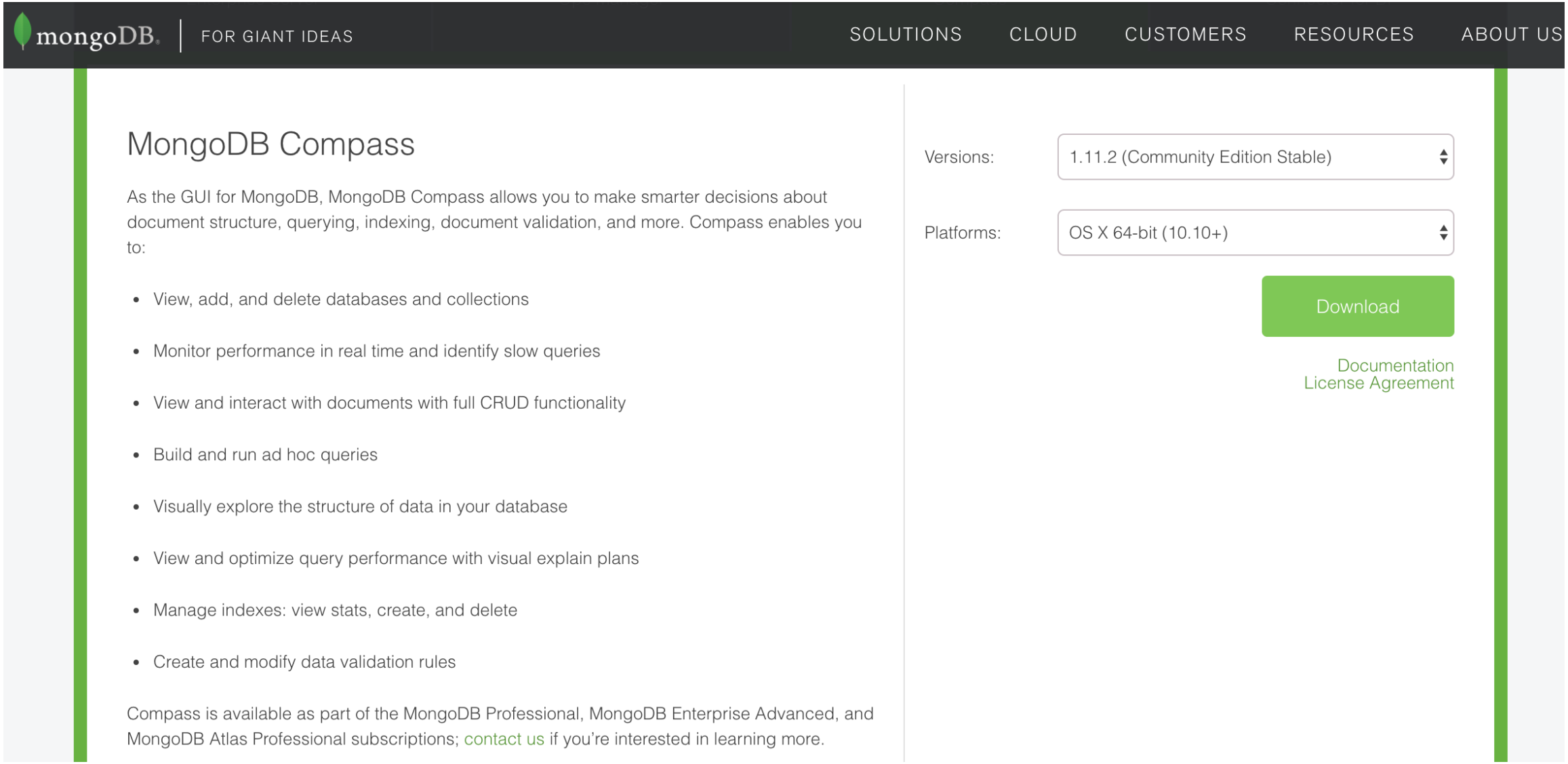
Install on macOS

Install MongoDB Community Edition on macOS systems from Homebrew packages or from MongoDB archives.

Install on Windows

Install MongoDB Community Edition on Windows systems and optionally start MongoDB as a Windows service.

MongoDB Compass



Running MongoDB

Add $MONGO_HOME to your path

```
export APPS=/Users/rachlin/apps
export MONGO_HOME=$APPS/mongo/mongodb-4.0.7
export PATH=$MONGO_HOME/bin:$PATH
```

Set up a directory for the database storage and run server

```
$ mkdir -p $HOME/data/mongodb
$ mongod --dbpath $HOME/data/mongodb
```

Run the command-line client (or launch Compass)

```
$ mongo
```