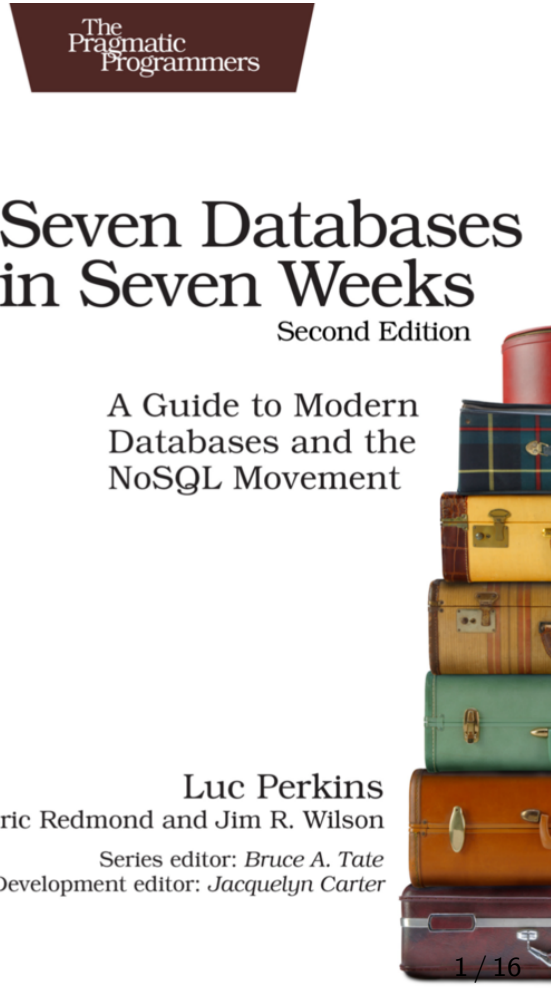
MongoDB and CouchDB

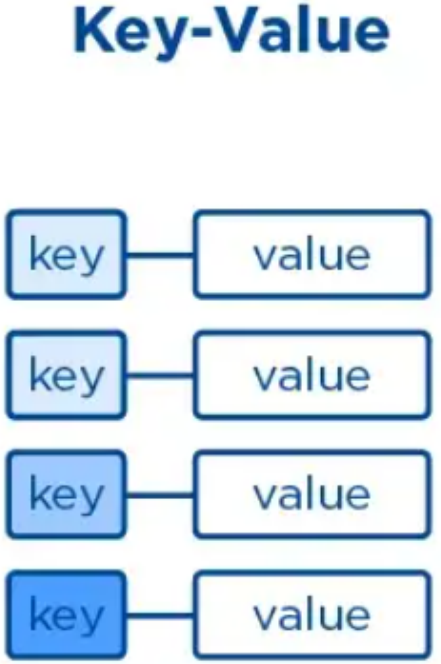
Instructor: Dr. GP Saggese - gsaggese@umd.edu

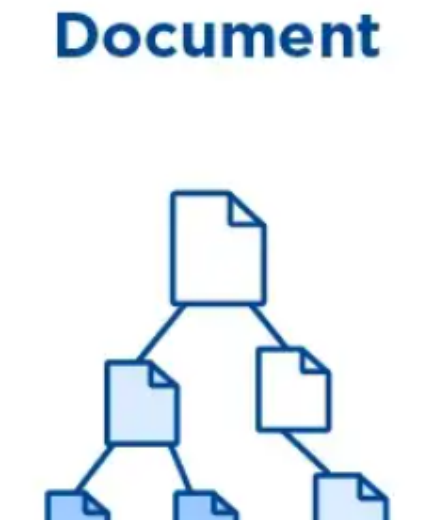
- All concepts in slides
- MongoDB tutorial
- Web
  - https://www.mongodb.com/
  - Official docs
  - pymongo
- Book
  - Seven Databases in Seven Weeks, 2e



Key-Value Store vs Document DBs

- **Key-value stores**
  - Basically a map or a dictionary
    - E.g., HBase, Redis
  - Typically only look up values by key
    - Sometimes can do search in value field with a pattern
  - Uninterpreted value (e.g., binary blob) associated with a key
  - Typically one namespace for all key-values
- **Document DBs**
  - Collect sets of key-value pairs into *documents*
    - E.g., MongoDB, CouchDB
  - Documents represented in JSON, XML, or BSON (binary JSON)
  - Documents organized into *collections*
    - Similar to *tables* in relational DBs
    - Large collections can be partitioned and indexed





MongoDB

- Developed by MongoDB Inc.
  - Founded in 2007
  - Based on DoubleClick experience with large-scale data
  - Mongo comes from "hu-mongo-us"
- One of the most used NoSQL DBs (if not the most used)
- **Document-oriented NoSQL database**
  - Schema-less
    - No Data Definition Language (DDL), like for SQL
    - You can store maps with any keys and values
    - Application tracks the schema, mapping between documents and their meaning
  - Keys are hashes stored as strings
    - Document Identifiers **_id** created for each document (field name reserved by Mongo)
  - Values use BSON format
    - Based on JSON (B stands for Binary)
- Written in C++
- Supports APIs (drivers) in many languages
  - E.g., JavaScript, Python, Ruby, Java, Scala, C++, ...

MongoDB: Example of Document

- **A document is a JSON data structure**
- It corresponds to a row in a relational DB
  - Without schema
  - Primary key is `_id`
  - Values nested to an arbitrary depth

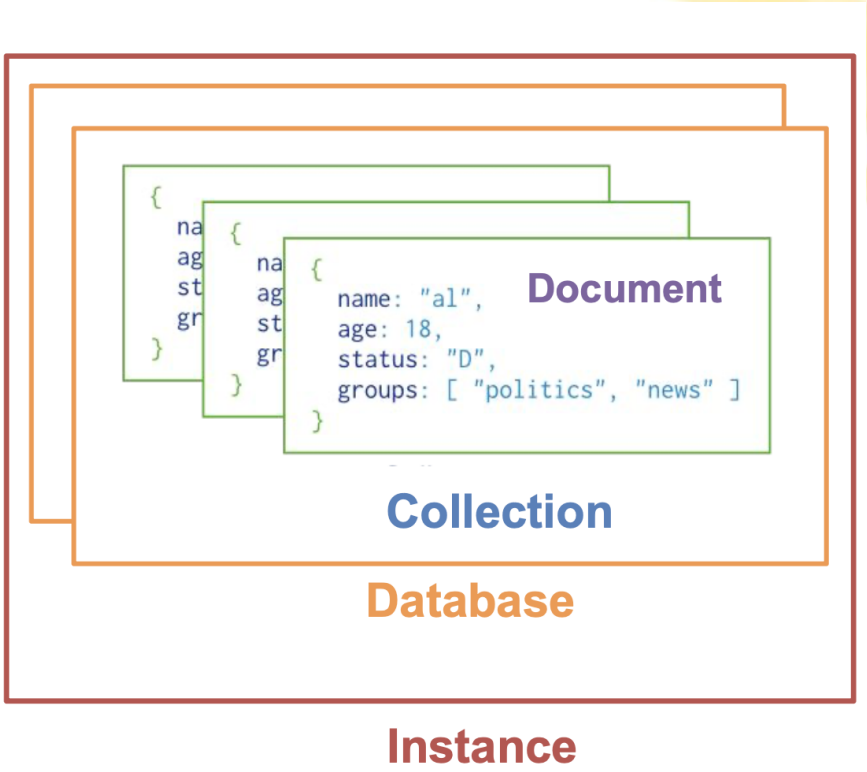
```
{
   "_id" : ObjectId("4d0b6da3bb30773266f39fea"),
   "country" : {
      "$ref" : "countries",
      "$id" : ObjectId("4d0e6074deb8995216a8309e")
   },
   "famous_for" : [
      "beer",
      "food"
   ],
   "last_census" : "Sun Jan 07 2018 00:00:00 GMT -0700 (PDT)",
   "mayor" : {
      "name" : "Ted Wheeler",
      "party" : "D"
   },
   "name" : "Portland",
   "population" : 582000,
   "state" : "OR"
}
```

MongoDB: Functionalities

- **Design goals**
  - Performance
  - Availability / scalability
  - Rich data storage (not rich querying!)
- **Dynamic schema**
  - No DDL (Data Definition Language)
  - Secondary indexes
  - Query language via an API
- **Several levels of data consistency**
  - E.g., atomic writes and fully-consistent reads (at document level)
- **No joins nor transactions across multiple documents**
  - Makes distributed queries easy and fast
- **High availability through replica sets**
  - E.g., primary replication with automated failover
- **Built-in sharding**
  - Horizontal scaling via automated range-based partitioning of data
  - Reads and writes distributed over shards

MongoDB: Hierarchical Objects

- A Mongo **instance** has:
  - Zero or more "databases"
  - Mongo instance ~ Postgres instance
- A Mongo **database** has:
  - Zero or more "collections"
    - Mongo collection ~ Postgres tables
  - Mongo database ~ Postgres database
- A Mongo **collection** has:
  - Zero or more "documents"
    - Mongo document ~ Postgres rows
- A Mongo **document** has:
  - One or more "fields"
    - It has always primary key `_id`
    - Mongo field ~ Postgres columns



ref

SCIENCE ACADEMY

Relational DBs vs MongoDB: Concepts

RDBMS Concept	MongoDB Concept	Meaning in MongoDB
database	database	Container for collections
relation / table / view	collection	Group of documents
row / instance	document	Group of fields
column / attribute	field	A name-value pair
index	index	Automatic
primary keys	_id field	Always the primary key
foreign key	reference	Pointers
table joins	embedded documents	Nested name-value pairs

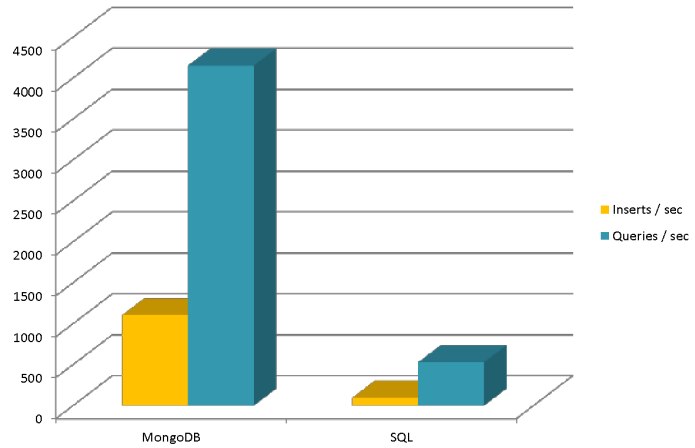
```
{
   "_id" : ObjectId("4d0b6da3bb30773266f39fea"),
   "country" : {
      "$ref" : "countries",
      "$id" : ObjectId("4d0e6074deb8995216a8309e")
   },
   "famous_for" : [
      "beer",
      "food"
   ],
   "last_census" : "Sun Jan 07 2018 00:00:00 GMT -0700 (PDT)",
   "mayor" : {
      "name" : "Ted Wheeler",
      "party" : "D"
   },
   "name" : "Portland",
   "population" : 582000,
   "state" : "OR"
}
```

Relational vs Document DB: Workflows

- **Relational DBs**
  - E.g., PostgreSQL
  - Know what you want to store
    - Tabular data
  - Do not know how to use it
    - Static schema allows query flexibility (e.g., joins)
  - Complexity is at insertion time
    - Decide how to represent the data (i.e., schema)
- **Document DBs**
  - E.g., MongoDB
  - No assumptions on what to store
    - E.g., irregular JSON data
  - Know a bit how to access data
    - You want to access the data by key
    - E.g., it's a nested key-value map
  - Complexity is at access time
    - Get the data from the server
    - Process data on the client side

Why Use MongoDB?

- Simple to query
  - Do the work on client side
- It's fast
  - 2-10x faster than Postgres
- Data model / functionalities suitable for most web applications
  - Semi-structured data
  - Quickly evolving systems
- Easy and fast integration of data
- Not well suited for heavy and complex transactions systems
  - E.g., banking system



MongoDB: Data Model

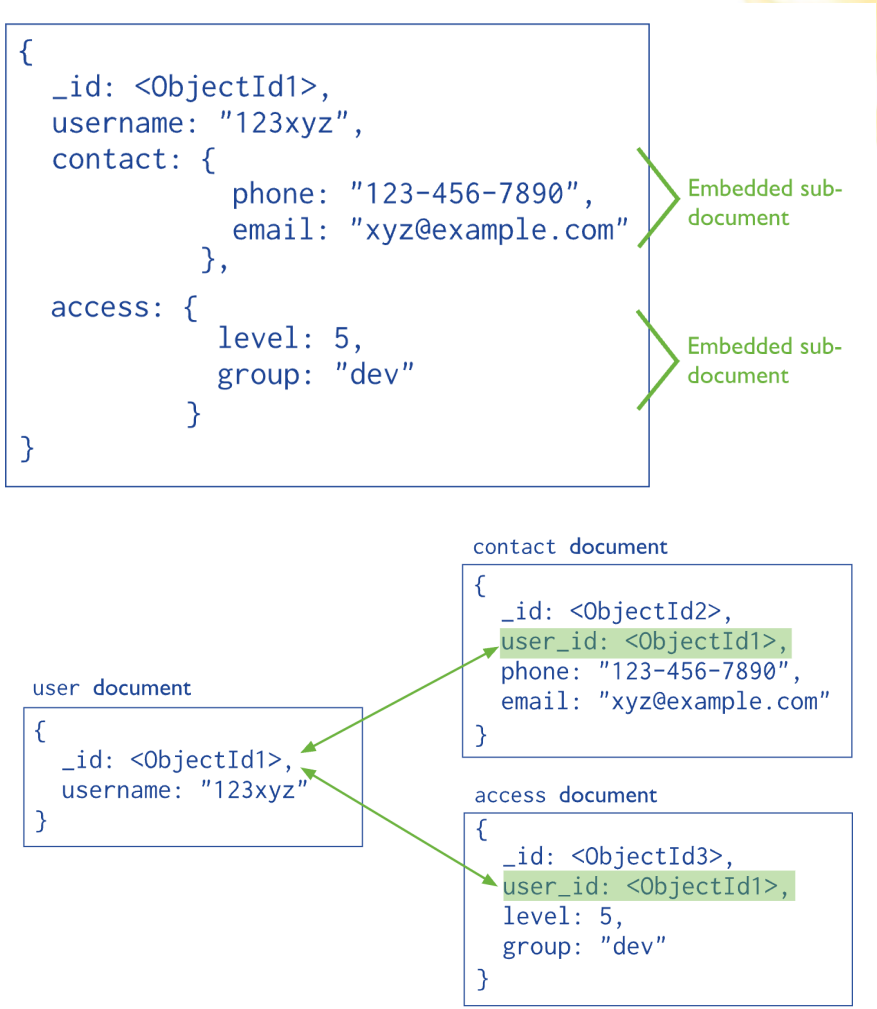
- **Documents** are composed of field and value pairs
  - **Field names** are strings
  - **Values** are any BSON type
    - Arrays of documents
    - Native data types
    - Other documents
- E.g.,
  - `_id` holds an `ObjectId`
  - name holds a document that contains the fields first and last
  - birth and death are of Date type
  - contribs holds an array of strings
  - views holds a value of the NumberLong type

```
{
   name: "sue",                          <-- field: value
   age: 26,                              <-- field: value
   status: "A",                          <-- field: value
   groups: [ "news", "sports" ]          <-- field: value
}
```

```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),
   name: { first: "Alan", last: "Turing" },
   birth: new Date('Jun 23, 1912'),
   death: new Date('Jun 07, 1954'),
   contribs: [ "Turing machine", "Turing test", "Turingery" ],
   views : NumberLong(1250000)
}
```

MongoDB: Data Model

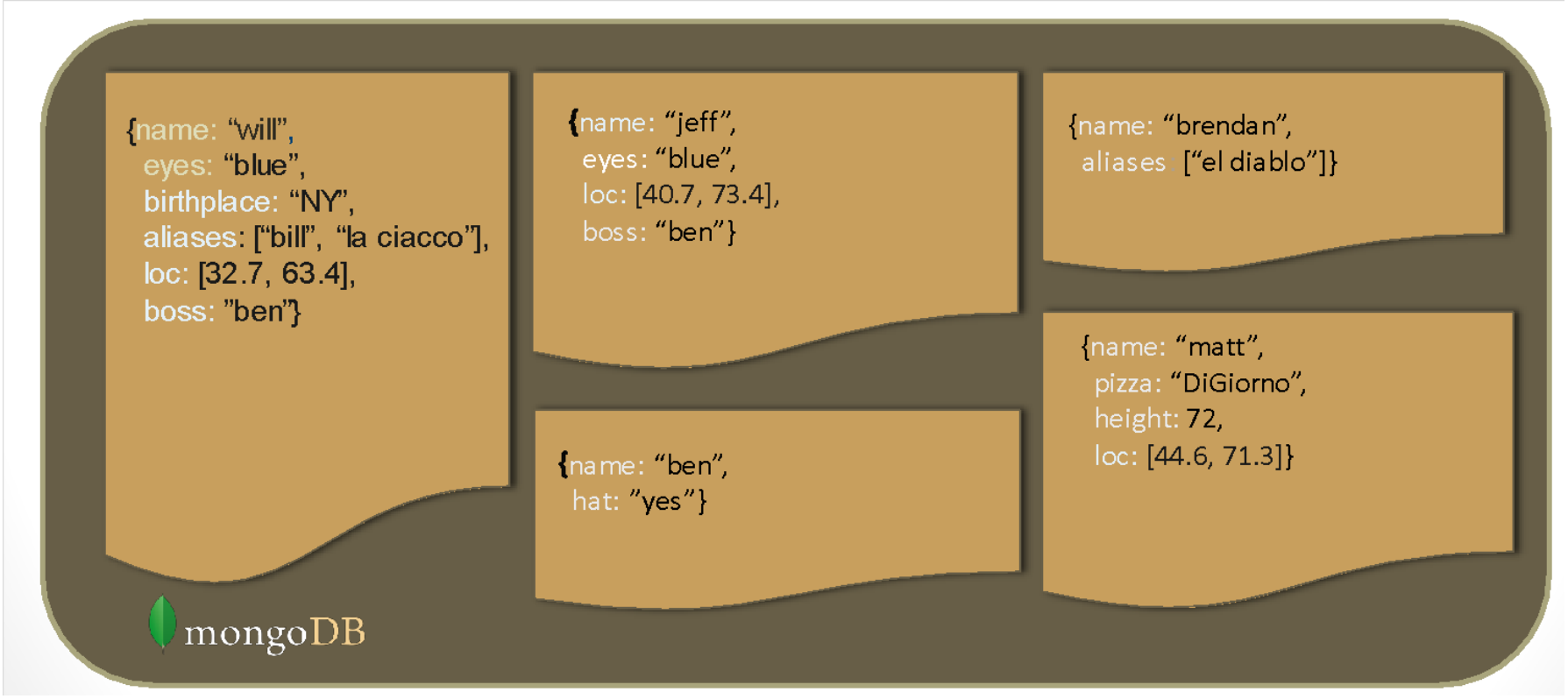
- Documents can be nested
  - Embedded sub-document
- **Denormalized data models**
  - Store multiple related pieces of information in the same record
  - Conceptually is the result of a join operation
- **Normalized data models**
  - Eliminate duplication
  - Represent many-to-many relationships



Schema Free

- MongoDB does not need any pre-defined data schema
- Every **document** in a **collection** can have different fields and values
  - No need for `NULL` values / union of fields like in relational DBs
- E.g., dishomogeneous data instances

Document **Document** **Collection**



JSON Format

- JSON = JavaScript Object Notation
- Data is stored in field / value pairs
- A field / value pair consists of:
  - A field name (always a string)
  - Followed by a colon :
  - Followed by a typed value

```
"name": "R2-D2"
```

- **Data in documents is separated by commas ‘,‘**

```
"name": "R2-D2", race: "Droid"
```

- Curly braces {} hold documents

```
{"name": "R2-D2", race : "Droid", affiliation: "rebels"}
```

- An array is stored in brackets []

```
[{"name": "R2-D2", race: "Droid", affiliation: "rebels"},
 {"name": "Yoda", affiliation: "rebels"}]
```

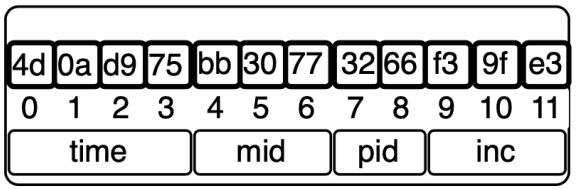
- Supports:

BSON Format

- Binary-encoded serialization of JSON-like documents
  - https://bsonspec.org
- Zero or more key/value pairs are stored as a single entity
  - Each entry consists of:
    - a field name (string)
    - a data type
    - a value
- Similar to Protocol Buffer, but more schema-less
- Large elements in a BSON document are prefixed with a length field to facilitate scanning
- MongoDB understands the internals of BSON objects, even nested ones
  - Can build indexes and match objects against query expressions for BSON keys

ObjectID

- Each JSON data contains an `_id` field of type `ObjectId`
  - Same as a SERIAL constraint incrementing a numeric primary key in PostgreSQL
- An `ObjectId` is 12 bytes, composed of:
  - a timestamp
  - client machine ID
  - client process ID
  - a 3-byte auto-incremented counter
- Each Mongo process can handle its own ID generation without colliding
  - Mongo has a distributed nature
- Details here



Indexes

- **Primary index**
  - Automatically created on the _id field
  - B+ tree indexes
- **Secondary index**
- Users can create secondary indexes to:
  - Improve query performance
  - Enforce unique values for a particular field
- Single field index and compound index (like SQL)
  - Order of the fields in a compound index matters
- Sparse property of an index
  - The index contains only entries for documents that have the indexed field
  - Ignore records that do not have the field defined
- Reject records with duplicate key value if an index is unique and sparse
- Details here