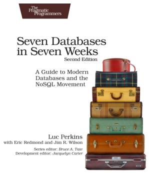
6.1: MongoDB

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

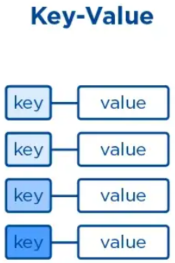
- **References**:
  - All concepts in slides
  - MongoDB tutorial
  - Web
    - https://www.mongodb.com/
    - Official docs
  - Seven Databases in Seven Weeks, 2e

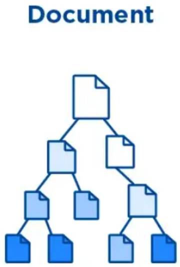

The Pragmatic Programmers
Seven Databases in Seven Weeks
Second Edition
A Guide to Modern Databases and the NoSQL Movement
Luc Perkins
with Eric Redmond and Jim R. Wilson
Series editor: Bruce A. Tate
Development editor: Jacquelyn Carter

SCIENCE ACADEMY

Key-Value Store vs Document DBs

- **Key-value stores**
  - Function as a map or dictionary
    - Examples: HBase, Redis
  - Primarily retrieve values using keys
  - Occasionally search within value fields using patterns
  - Store uninterpreted values (e.g., binary blobs) linked to keys
  - Use a single namespace for all key-value pairs
- **Document DBs**
  - Group key-value pairs into *documents*
    - Examples: MongoDB, CouchDB
  - Documents formatted in JSON, XML, or BSON (binary JSON)
  - Documents are part of *collections*
    - Comparable to *tables* in relational databases
  - Large collections can be partitioned and indexed





SCIENCE ACADEMY

MongoDB

- **Developed by MongoDB Inc**
  - Founded in 2007
  - Based on DoubleClick's experience with large-scale data
  - Mongo comes from "hu-mongo-us"
- **Highly popular NoSQL database**
- **Document-oriented NoSQL DB**
  - Schema-less
    - No Data Definition Language (DDL) like SQL
    - Stores maps with any keys and values
    - Application manages schema, linking documents to meanings
  - Keys are string-stored hashes
    - Each document has a unique `_id` (reserved by MongoDB)
  - Values in BSON format
    - Based on JSON (B for Binary)
- **High-performance**
  - Developed in C++
  - Supports APIs (drivers) in various languages
    - Examples: JavaScript, Python, Ruby, Java, Scala, C++, etc

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 can be 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"
}
```

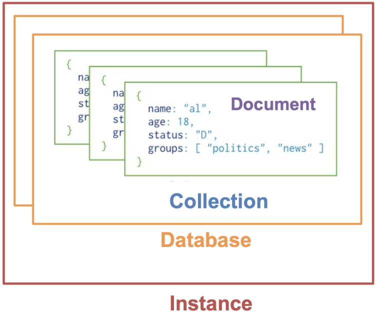
SCIENCE ACADEMY

MongoDB: Functionalities

- **Design goals**
  - Performance
  - Availability/scalability
  - Rich data storage (not rich querying!)
- **Dynamic schema**
  - No DDL
  - Secondary indexes
  - Query language via API
- **Several levels of data consistency**
  - Atomic writes and fully-consistent reads (document level)
- **No joins nor transactions across multiple documents**
  - Distributed queries easy and fast
- **High availability through replica sets**
  - Primary replication with automated failover
- **Built-in sharding**
  - Horizontal scaling via automated range-based partitioning
  - 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”
    - Always has a primary key `_id`
    - Mongo field ~ Postgres columns



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

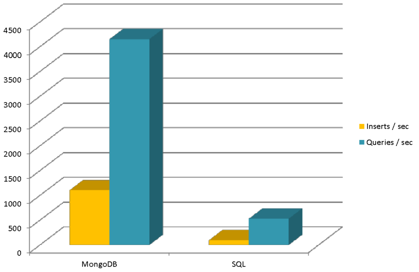
```
{
   "_id" : ObjectId("4d0b6da3bb30773266f39fea"),
   "country" : {
      "$ref" : "countries",
      "$id" : ObjectId("4d0e6074deb8995216a8309e")
   },
   "famous_for" : [
      "beer",
      "food"
   ],
   "last_census" : "Sun Jan 07 2018 00:00:00 GMT -0700 (PDT)",
   "mayor" : {
      "name"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

Relational vs Document DB: Workflows

- Relational DBs
  - E.g., PostgreSQL
  - Know what to store
    - Tabular data
  - Static schema allows query flexibility (e.g., joins)
  - Complexity at insertion time
    - Decide data representation (schema)
- Document DBs
  - E.g., MongoDB
  - No assumptions on storage
    - E.g., irregular JSON data
  - Access data by key
    - Nested key-value map
  - Complexity at access time
    - Retrieve data from server
    - Process data client-side

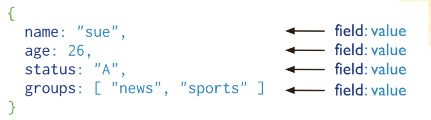
Why Use MongoDB?

- Simple and powerful to query
- Fast
  - 2-10x faster than PostgreSQL
- Data model suitable for most web applications
  - Semi-structured data
  - Quickly evolving systems
- Not suited for heavy, complex transaction systems
  - E.g., banking systems

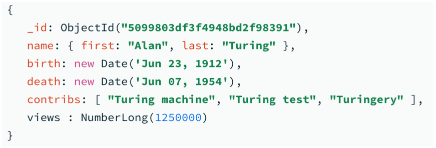


MongoDB: Data Model

- **Documents** are field-value pairs
  - *Field names*: strings
  - *Values*: any BSON type
    - Arrays of documents
    - Native data types
    - Other documents
- **Examples**:
  - `_id`: `ObjectId`
  - `name`: document with fields `first` and `last`
  - `birth` and `death`: date type
  - `contribs`: array of strings
  - `views`: `NumberLong` type



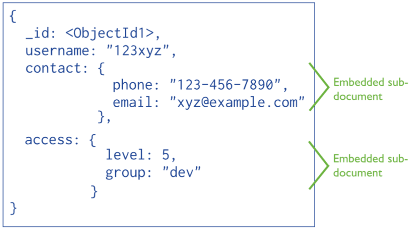
```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```



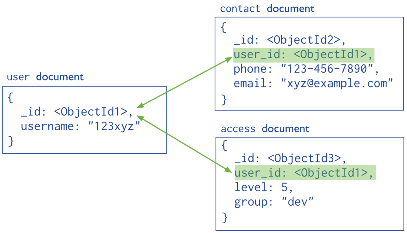
```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),
   name: { first: "Alan", last: "Turing" },
   birth: new Date('Jun 23, 1912'),
   death: new Date('Jun 07, 1954'),
   contribs: [ "Turing machine", "Turing test", "Turingery" ],
   views : NumberLong(1250000)
}
```

MongoDB: Data Model

- **Documents can be nested**
  - Embedded sub-document
- **Denormalized data models**
  - Store related information in the same record
  - Avoids the need for a join operation
- **Normalized data models**
  - Eliminate duplication
  - Represent many-to-many relationships



```
{
  _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

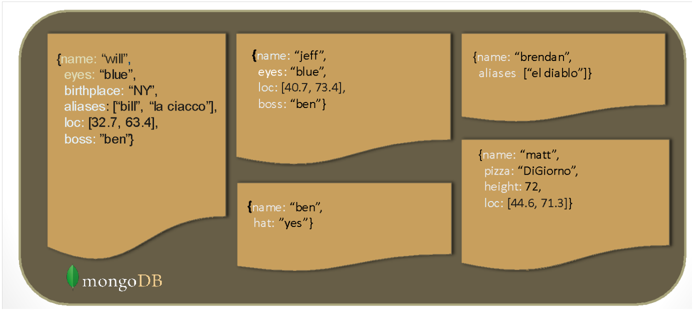
```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

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

Schema Free

- MongoDB does not need 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., heterogeneous data instances



```
{name: "will",
  eyes: "blue",
  birthplace: "NY",
  aliases: ["bill", "la ciacco"],
  loc: [32.7, 63.4],
  boss: "ben"}
```

```
{name: "jeff",
  eyes: "blue",
  loc: [40.7, 73.4],
  boss: "ben"}
```

```
{name: "brendan",
  aliases ["el diablo"]}
```

```
{name: "ben",
  hat: "yes"}
```

```
{name: "matt",
  pizza: "DiGiorno",
  height: 72,
  loc: [44.6, 71.3]}
```

SCIENCE ACADEMY

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"
  ```
  - Just references

- 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 []

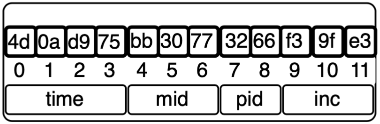
  ```
  [
    { "name": "R2-D2", "race": "Droid", "affiliation": "rebels" },
    { "name": "Yoda", "affiliation": "rebels" }
  ]
  ```

BSON Format

- **Binary-encoded serialization of JSON-like documents**
    - https://bsonspec.org
    - Similar to Protocol Buffers, but more schema-less
- **Optimized for random access**
    - Prefixes elements in a BSON document with a length field
- **MongoDB understands BSON objects, even nested ones**
    - Builds indexes and matches objects against query expressions for BSON keys

ObjectID

- Each JSON document contains an `_id` field of type `ObjectId`
  - Similar to the SERIAL constraint incrementing a numeric primary key in PostgreSQL
- An `ObjectId` is 12 bytes, composed of:
  - Timestamp
  - Client machine ID
  - Client process ID
  - 3-byte auto-incremented counter



- Each MongoDB process handles its own ID generation without collisions
  - Due to MongoDB's distributed nature
- Details here

SCIENCE ACADEMY

Indexes

- **Primary index**
  - Automatically created on the `_id` field
  - B+ tree indexes
- **Secondary index**
  - Improve query performance
  - Can enforce unique values for a field
- Single field index and compound index (like SQL)
  - Order of fields in a compound index matters
- Sparse property of an index
  - Index contains entries only for documents with the indexed field
  - Ignores records without the field
- Rejects records with duplicate keys if the index is unique and sparse
- Details here

SCIENCE ACADEMY