Document Databases

Chapter 15

Michael Tsiang

Stats 167: Introduction to Databases

UCLA

Document Databases

MongoDB

Document Databases

Extending the Key-Value Database Model

By having no schema or imposed structure, key-value databases excel at speed and flexibility. The values stored in a key-value database can be nearly any type of object.

However, key-value databases store values as whole data objects (or blobs) that cannot be queried or indexed, which can limit the types of queries sent to the database.

What if we want to store and query more complex or nested data? For example, how can we store user profiles with lists and embedded objects that also can be queried?

Document Databases

A **document database** extends the key-value model by storing values as JSON-like objects called **documents** that can be structured and queried.

A group of related documents is called a **collection** (analogous to a table in a relational database).

Some characteristics of document databases:

- JSON-like: Most document databases use JSON, BSON (binary JSON), XML, or other similar format.
- Flexible structure (**schema-on-read**): Each document in a collection can have different structure.
- Nested: Documents can contain arrays and nested (or **embedded**) documents.
- Queryable: Fields (e.g., `email`) and nested fields (e.g., `favorites.food`) can be indexed and filtered in queries.

Example Document and Query

An example of a user profile document (in JSON format) with nested and array fields:

```
{
  "username": "leslie_knope",
  "age": 36,
  "email": "leslie@pawnee.gov",
  "favorites": {
    "color": "green",
    "food": "waffles",
    "animal": "mini horses"
  },
  "hobbies": ["working", "bowling"]
}
```

An example query (in MongoDB):

```
db.users.find({ "favorites.food": "waffles" })
```

Document Database Use Cases

Some examples of use cases for document databases:

- Event logs
- Content management systems
- Product catalogs
- User profiles and preferences
- Real-time analytics
- Mobile applications (e.g., Uber, Tinder)
- Video game metrics

MongoDB customer case studies:
https://www.mongodb.com/solutions/customer-case-studies

When Not to Use

Document databases are useful for scenarios with hierarchical or nested semi-structured data. However, there are also scenarios in which a document structure is not ideal.

Some scenarios in which a document database would not be suitable:

- Relational data

  Document databases do not support joins or enforce foreign key constraints, so they do not ensure referential integrity.

- A fixed schema for enforcing structure and constraints

  Document databases allow documents in the same collection to have different structures.

- Complex transactions across many documents

  Some document databases support multi-document transactions, but they are typically less efficient and more limited than those in relational databases.

MongoDB

What is MongoDB?

MongoDB (https://www.mongodb.com/) is the most popular document database.

MongoDB stores data in BSON (Binary JSON), a binary-encoded format for JSON-like objects that is designed to be more efficient for storing and transmitting data.

Documents are stored in field-value pairs. Field names are strings, but values can be any of the supported BSON data types: https://www.mongodb.com/docs/manual/reference/bson-types/

MongoDB Document Structure

An example of a MongoDB document:

```
{
  "_id": ObjectId("5099803df3f4948bd2f98391"),
  "name": "Ron Swanson",
  "age": 45,
  "department": "Parks and Recreation",
  "skills": ["woodworking", "hunting", "grilling"],
  "address": {
    "city": "Pawnee",
    "state": "Indiana"
  }
}
```

The field `_id` serves as a primary key and is required in every MongoDB document. It can be user-specified or it will be automatically generated.

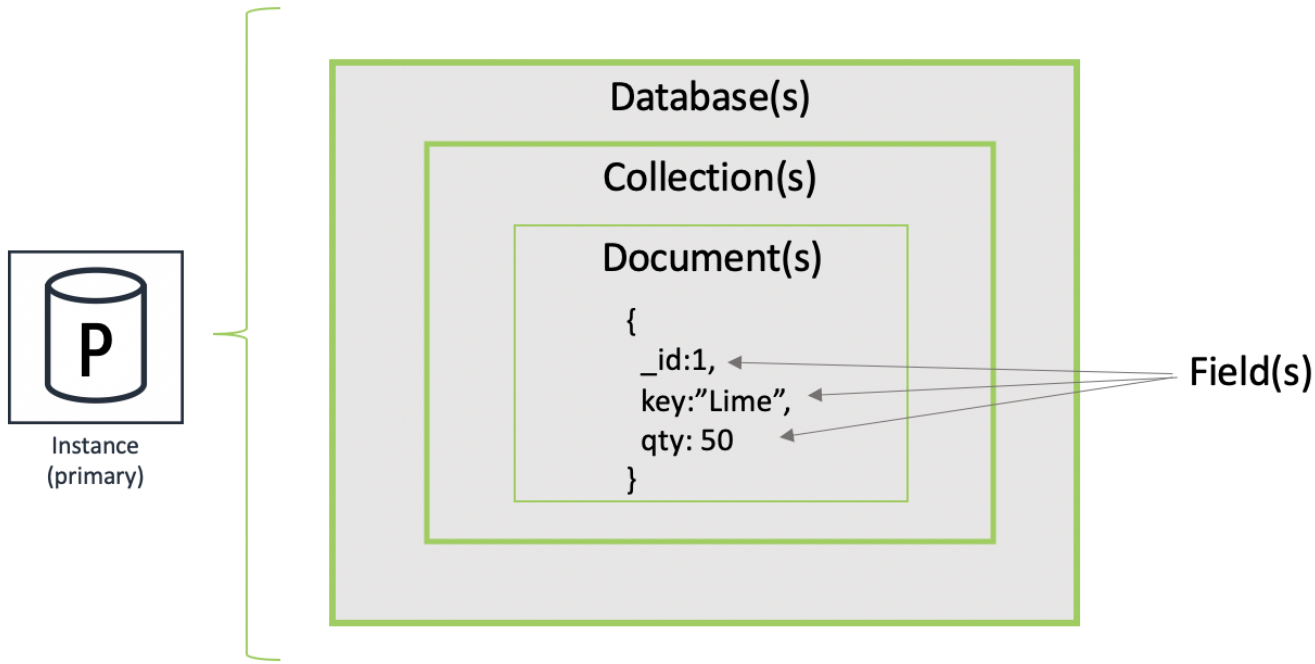
MongoDB uses **dot notation** to access fields within embedded documents and elements within arrays, using names for nested values and zero-based indexes for array elements. For example, `"address.city"` accesses a nested field and `"skills.0"` accesses the first item in an array.

MongoDB Data Model

While each MongoDB document has a hierarchical structure, the overall MongoDB data model is also hierarchical.

MongoDB stores data records as BSON documents, which are grouped into collections. A **database** stores one or more collections of documents.

An illustration of this hierarchy (database → collection → document) is shown below:

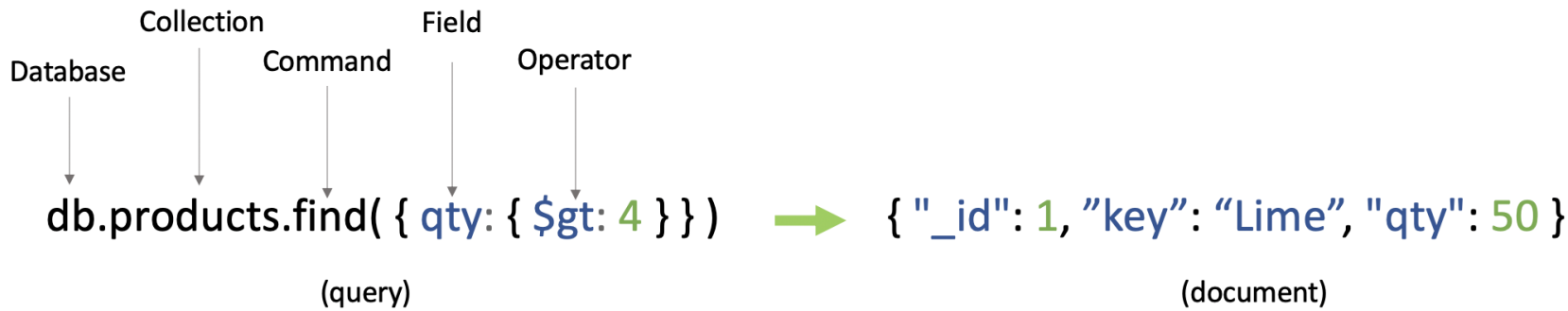


Source: AWS article, "What Is a Document Database?"

MongoDB Query Structure

MongoDB queries are written in **MQL (MongoDB Query Language)**, which uses JSON-like syntax.

Queries in MQL follow a standard structure, as shown below:



Source: AWS article, "What Is a Document Database?"

MongoDB CRUD Operations

In MongoDB, operations on documents are performed using methods that act on collections.

The general syntax is **`db.collection.method()`**, where `db` refers to the current database, and `collection` is the name of the collection (e.g., `db.employees.find()`).

The four core CRUD operations are:

- **`insertOne()`**: Insert a single document
- **`find()`**: Retrieve documents that match a filter (returns a cursor)
- **`updateOne()`**: Update the first matching document
- **`deleteOne()`**: Delete the first matching document

For bulk operations, use **`insertMany()`**, **`updateMany()`**, and **`deleteMany()`**. For more complex or mixed combinations, they can all be combined using the **`bulkWrite()`** method.

Examples

Example CRUD operations on the `employees` collection:

```
// Create
db.employees.insertOne({ name: "Ron", age: 45 })

// Read
db.employees.find({ age: 45 })

// Update
db.employees.updateOne(
  { name: "Ron" },
  { $set: { age: 46 } }
)

// Delete
db.employees.deleteOne({ name: "Ron" })
```

Getting Started with MongoDB

There are a few ways to get started with trying MongoDB for yourself.

- **MongoDB Atlas** is fully-managed cloud database (DBaaS) with a free tier:
  https://www.mongodb.com/basics/mongodb-atlas-tutorial
- MongoDB also offers a free Community edition, which you can download locally here:
  https://www.mongodb.com/try/download/community
- Try it online with the Mongo Playground (no installation):
  https://mongoplayground.net/

MQL Operators

MongoDB supports many operators that enable complex queries.

Some common operators:

- Equality and comparison: `$eq`, `$gt`, `$gte`, `$lt`, `$lte`
- Array queries: `$in`, `$all`
- Logical: `$and`, `$or`, `$not`
- Element checks: `$exists`, `$type`

For example:

```
db.employees.find({
  age: {$gte: 30},
  "skills": {$in: ["claymation", "accounting"]}
})
```

MongoDB Aggregation Pipelines

An **aggregation pipeline** is a sequence of operations that process, transform, and return data. Aggregation pipelines enable building complex queries step by step, combining stages like filtering, grouping, and sorting to analyze data efficiently without moving it out of the database.

A typical pipeline:

Input → `$match` → `$group` → `$sort` → Output

This is roughly equivalent to the following SQL flow:

Input → `WHERE` → `GROUP BY` → `ORDER BY` → Output

For example:

```
db.orders.aggregate([
  {$match: {status: "shipped"}},
  {$group: {
    _id: "$customer_id",
    total: {$sum: "$amount"}
  }},
  {$sort: {total: -1}}
])
```

Connecting to MongoDB with R and Python

MongoDB supports multiple APIs to connect with its databases. Their standard APIs are MongoShell (a command line interface), Java, Javascript, and Python, but there is also an R package for MongoDB too.

In particular:

- The `pymongo` library in Python
- The `mongolite` package in R

MongoDB with Python

Example using the `pymongo` library to insert and retrieve a document:

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["parks_and_rec"]
collection = db["employees"]

collection.insert_one({
  "name": "Andy",
  "role": "Shoeshiner"
})
collection.find_one({"name": "Andy"})
```

MongoDB with R

Example using the `mongolite` package to connect, insert, and find a document:

```
library(mongolite)

m <- mongo(collection = "employees",
           db = "parks_and_rec",
           url = "mongodb://localhost")

m$insert('{"name": "Donna", "age": 38}')
m$find('{"name": "Donna"}')
```

Further Learning

For more in-depth information on MongoDB:

- MongoDB Documentation: https://www.mongodb.com/docs/
- MongoDB for SQL Experts: https://learn.mongodb.com/courses/mongodb-for-sql-experts
- MongoDB Learning Tutorials: https://learn.mongodb.com/

A tutorial on aggregation pipelines:

- The Beginner's Guide to MongoDB Aggregation: https://studio3t.com/knowledge-base/articles/mongodb-aggregation-framework/