NoSQL Databases

Advanced Topics in Databases

NoSQL

- NoSQL databases are currently a hot topic in some parts of computing, with over a hundred different NoSQL databases.

RDBMS Characteristics

- Data stored in columns and tables
- Relationships represented by data
- Data Manipulation Language
- Data Definition Language
- Transactions
- Abstraction from physical layer
- Applications specify what, not how
- Physical layer can change without modifying applications
    - Create indexes to support queries
    - In Memory databases

Transactions – ACID Properties

- Atomic – All of the work in a transaction completes (commit) or none of it completes
  - a transaction to transfer funds from one account to another involves making a withdrawal operation from the first account and a deposit operation on the second. If the deposit operation failed, you don't want the withdrawal operation to happen either.
- Consistent – A transaction transforms the database from one consistent state to another consistent state. Consistency is defined in terms of constraints.
  - a database tracking a checking account may only allow unique check numbers to exist for each transaction
- Isolated – The results of any changes made during a transaction are not visible until the transaction has committed.
  - a teller looking up a balance must be isolated from a concurrent transaction involving a withdrawal from the same account. Only when the withdrawal transaction commits successfully and the teller looks at the balance again will the new balance be reported.
- Durable – The results of a committed transaction survive failures
  - A system crash or any other failure must not be allowed to lose the results of a transaction or the contents of the database. Durability is often achieved through separate transaction logs that can "re-create" all transactions from some picked point in time (like a backup).

No SQL?

- NoSQL stands for:
  - No Relational
  - No RDBMS
  - Not Only SQL
- NoSQL is an umbrella term for all databases and data stores that don’t follow the RDBMS principles
  - A class of products
  - A collection of several (related) concepts about data storage and manipulation
  - Often related to large data sets

NoSQL Definition

From www.nosql-database.org:

> **Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge data amount, and more.**

Where does NoSQL come from?

- Non-relational DBMSs are not new
- But NoSQL represents a new incarnation
  - Due to massively scalable Internet applications
  - Based on distributed and parallel computing
- Development
  - Starts with Google
  - First research paper published in 2003
  - Continues also thanks to Lucene's developers/Apache (Hadoop) and Amazon (Dynamo)
  - Then a lot of products and interests came from Facebook, Netflix, Yahoo, eBay, Hulu, IBM, and many more

Dynamo and BigTable

- Three major papers were the seeds of the NoSQL movement
  - BigTable (Google)
  - Dynamo (Amazon)
    - Distributed key-value data store
    - Eventual consistency
  - CAP Theorem

NoSQL and Big Data

- NoSQL comes from Internet, thus it is often related to the “big data” concept
- How much big are “big data”?
  - Over few terabytes Enough to start spanning multiple storage units
- Challenges
  - Efficiently storing and accessing large amounts of data is difficult, even more considering fault tolerance and backups
  - Manipulating large data sets involves running immensely parallel processes
  - Managing continuously *evolving schema* and metadata for *semi-structured and un-structured* data is difficult

How did we get there?

- Explosion of social media sites (Facebook, Twitter) with large data needs
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution)
- Just as moving to dynamically-typed languages (Python, Ruby, Groovy), a shift to dynamically-typed data with frequent schema changes
- Open-source community

Why are RDBMS not suitable for Big Data

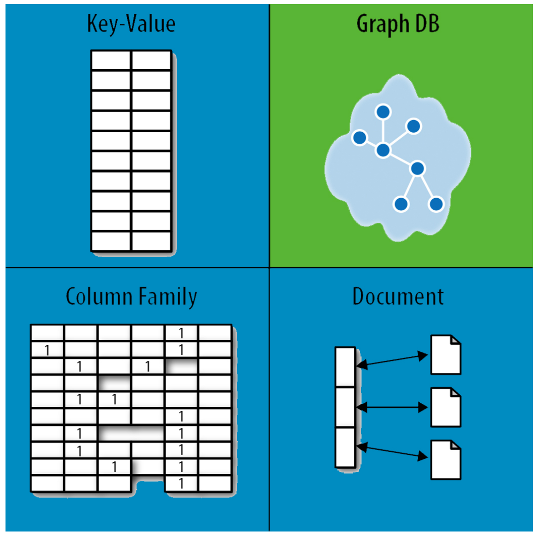
- The context is Internet
- RDBMSs assume that data are
    - Dense
    - Largely uniform (structured data)
- Data coming from Internet are
    - Massive and sparse
    - Semi-structured or unstructured
- With massive sparse data sets, the typical storage mechanisms and access methods get stretched

NoSQL Distinguishing Characteristics

- Large data volumes
  - Google's "big data"
- Scalable replication and distribution
  - Potentially thousands of machines
  - Potentially distributed around the world
- Queries need to return answers quickly
- Mostly query, few updates
- Asynchronous Inserts & Updates
- Schema-less
- ACID transaction properties are not needed – BASE
- CAP Theorem
- Open source development

NoSQL Database Types

Discussing NoSQL databases is complicated because there are a variety of types:



- Sorted ordered Column Store
  - Optimized for queries over large datasets, and store columns of data together, instead of rows
- Document databases:
  - pair each key with a complex data structure known as a document.
- Key-Value Store :
  - are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value.
- Graph Databases :
  - are used to store information about networks of data, such as social connections.

Document Databases (Document Store)

- Documents
  - Loosely structured sets of key/value pairs in documents, e.g., XML, JSON, BSON
  - Encapsulate and encode data in some standard formats or encodings
  - Are addressed in the database via a unique key
  - Documents are treated as a whole, avoiding splitting a document into its constituent name/value pairs
- Allow documents retrieving by keys or contents
- Notable for:
  - MongoDB (used in FourSquare, Github, and more)
  - CouchDB (used in Apple, BBC, Canonical, Cern, and more)

Document Databases (Document Store)

- The central concept is the notion of a "document“ which corresponds to a row in RDBMS.
- A document comes in some standard formats like JSON (BSON).
- Documents are addressed in the database via a unique *key* that represents that document.
- The database offers an API or query language that retrieves documents based on their contents.
- Documents are schema free, i.e., different documents can have structures and schema that differ from one another. (An RDBMS requires that each row contain the same columns.)

Document Databases, JSON

```
{
	_id: ObjectId("51156a1e056d6f966f268f81"),
	type: "Article",
	author: "Derick Rethans",
	title: "Introduction to Document Databases with MongoDB",
	date: ISODate("2013-04-24T16:26:31.911Z"),
	body: "This arti…"
},
{
	_id: ObjectId("51156a1e056d6f966f268f82"),
	type: "Book",
	author: "Derick Rethans",
	title: "php|architect's Guide to Date and Time Programming with PHP",
	isbn: "978-0-9738621-5-7"
}
```

Dealing with Big Data and Scalability

- Issues with scaling up when the dataset is just too big
- RDBMS were not designed to be distributed
- Traditional DBMSs are best designed to run well on a “single” machine
  - Larger volumes of data/operations requires to upgrade the server with faster CPUs or more memory known as ‘scaling up’ or ‘Vertical scaling’
- NoSQL solutions are designed to run on clusters or multi-node database solutions
  - Larger volumes of data/operations requires to add more machines to the cluster, Known as ‘scaling out’ or ‘horizontal scaling’
  - Different approaches include:
    - Master-slave
    - Sharding (partitioning)

Scaling RDBMS

- Master-Slave
  - All writes are written to the master. All reads performed against the replicated slave databases
  - Critical reads may be incorrect as writes may not have been propagated down
  - Large data sets can pose problems as master needs to duplicate data to
- Sharding
  - Any DB distributed across multiple machines needs to know in what machine a piece of data is stored or must be stored
  - A sharding system makes this decision for each row, using its key

NoSQL, No ACID

- RDBMSs are based on ACID (Atomicity, Consistency, Isolation, and Durability) properties
- NoSQL
  - Does not give importance to ACID properties
  - In some cases completely ignores them
- In distributed parallel systems it is difficult/impossible to ensure ACID properties
  - Long-running transactions don't work because keeping resources blocked for a long time is not practical

BASE Transactions

- Acronym contrived to be the opposite of ACID
  - **B**asically **A**vailable,
  - **S**oft state,
  - **E**ventually Consistent
- Characteristics
  - Weak consistency – stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler and faster

Performance

- There is no perfect NoSQL database
- Every database has its advantages and disadvantages
  - Depending on the type of tasks (and preferences) to accomplish
- NoSQL is a set of concepts, ideas, technologies, and software dealing with
  - Big data
  - Sparse un/semi-structured data
  - High horizontal scalability
  - Massive parallel processing
- Different applications, goals, targets, approaches need different NoSQL solutions

Where would I use it?

- Where would I use a NoSQL database?
- Do you have somewhere a large set of uncontrolled, unstructured, data that you are trying to fit into a RDBMS?
  - Log Analysis
  - Social Networking Feeds (many firms hooked in through Facebook or Twitter)
  - External feeds from partners
  - Data that is not easily analyzed in a RDBMS such as time-based data
  - Large data feeds that need to be massaged before entry into an RDBMS

Don’t forget about the DBA

- It does not matter if the data is deployed on a NoSQL platform instead of an RDBMS.
- Still need to address:
  - Backups & recovery
  - Capacity planning
  - Performance monitoring
  - Data integration
  - Tuning & optimization
- What happens when things don’t work as expected and nodes are out of sync or you have a data corruption occurring at 2am?
- Who you gonna call?
  - DBA and SysAdmin need to be on board

The Perfect Storm

- Large datasets, acceptance of alternatives, and dynamically-typed data has come together in a perfect storm
- Not a backlash/rebellion against RDBMS
- SQL is a rich query language that cannot be rivaled by the current list of NoSQL offerings
  - So you have reached a point where a read-only cache and write-based RDBMS isn’t delivering the throughput necessary to support a particular application.
  - You need to examine alternatives and what alternatives are out there.
  - The NoSQL databases are a pragmatic response to growing scale of databases and the falling prices of commodity hardware.

Summary

- Most likely, 10 years from now, the majority of data is still stored in RDBMS.
- Leading users of NoSQL datastores are social networking sites such as Twitter, Facebook, LinkedIn, and Digg.
- Not every problem is a nail and not every solution is a hammer.
- NoSQL has taken a field that was "dead" (database development) and suddenly brought it back to life.

MongoDB Tutorial

- https://www.w3schools.com/mongodb/index.php

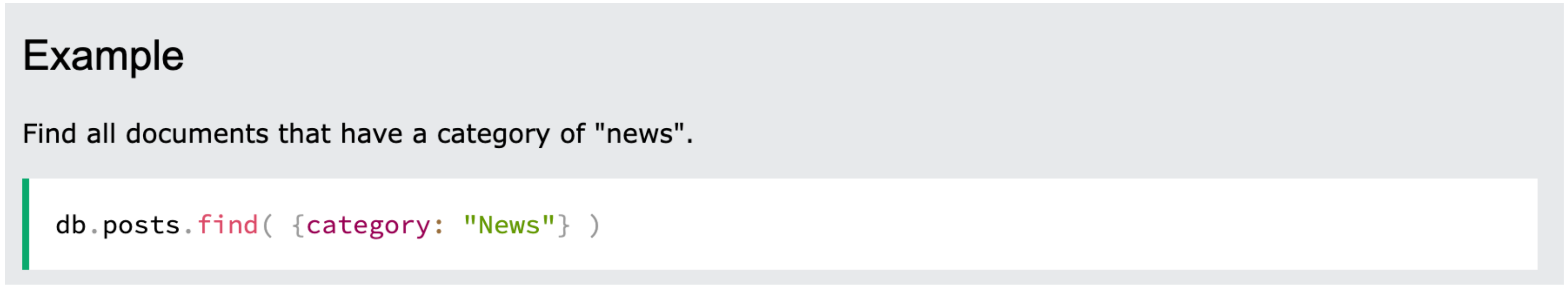
A MongoDB Document

- Records in a MongoDB database are called documents, and the field values may include numbers, strings, booleans, arrays, or even nested documents.

Example Document

```
{
        title: "Post Title 1",
        body: "Body of post.",
        category: "News",
        likes: 1,
        tags: ["news", "events"],
        date: Date()
}
```

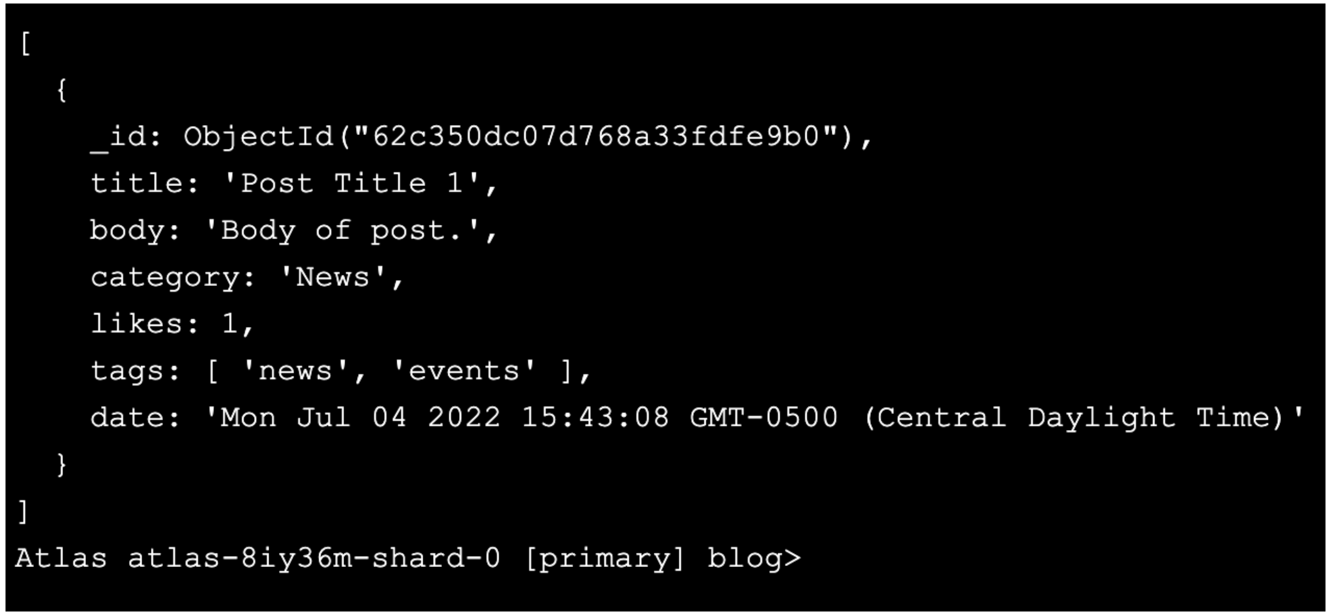
MongoDB Example



Example

Find all documents that have a category of "news".

```
db.posts.find( {category: "News"} )
```



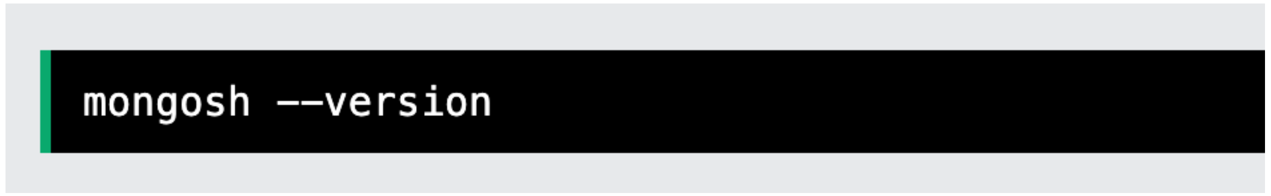
```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    body: 'Body of post.',
    category: 'News',
    likes: 1,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  }
]
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

Local vs Cloud Database

- MongoDB can be installed locally, which will allow you to host your own MongoDB server on your hardware. This requires you to manage your server, upgrades, and any other maintenance.
- You can download and use the MongoDB open source Community Server on your hardware for free.
- Here the outputs are from MongoDB Atlas, a cloud database platform. This is much easier than hosting your own local database.
- To be able to experiment with the code examples, you will need access to a MongoDB database.

Install MongoDB Shell (mongosh)

- There are many ways to connect to your MongoDB database.
- We will start by using the MongoDB Shell, mongosh.
- Use the official instructions to install mongosh on your operating system.
- To verify that it has been installed properly, open your terminal and type:



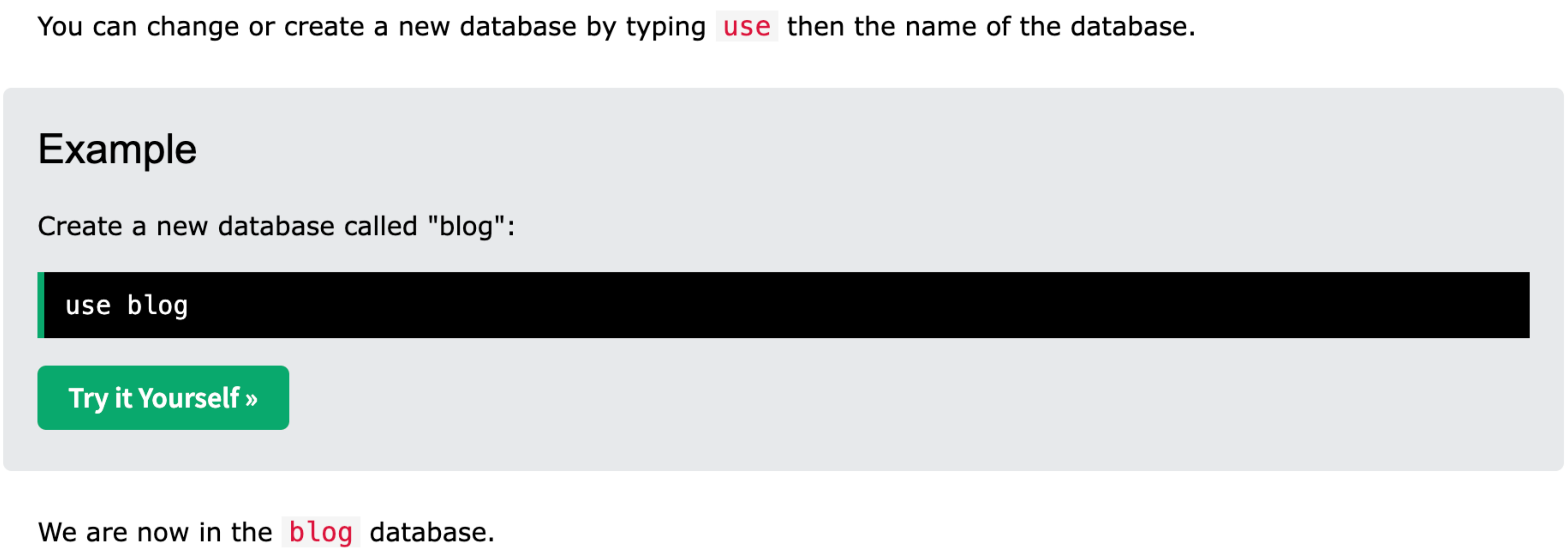
```
mongosh --version
```

- In the following slides we will use 'mongosh' to create, read, update, and delete (CRUD) items in a database.

MongoDB Query API

- The MongoDB Query API is the way you will interact with your data.
- The MongoDB Query API can be used two ways:
  - CRUD Operations
  - Aggregation Pipelines
- You can use the MongoDB Query API to perform:
  - Adhoc queries with mongosh, Compass, VS Code, or a MongoDB driver for the programming language you use.
  - Data transformations using aggregation pipelines.
  - Document join support to combine data from different collections.
  - Graph and geospatial queries.
  - Full-text search.
  - Indexing to improve MongoDB query performance.
  - Time series analysis.

Change or Create a Database



You can change or create a new database by typing `use` then the name of the database.

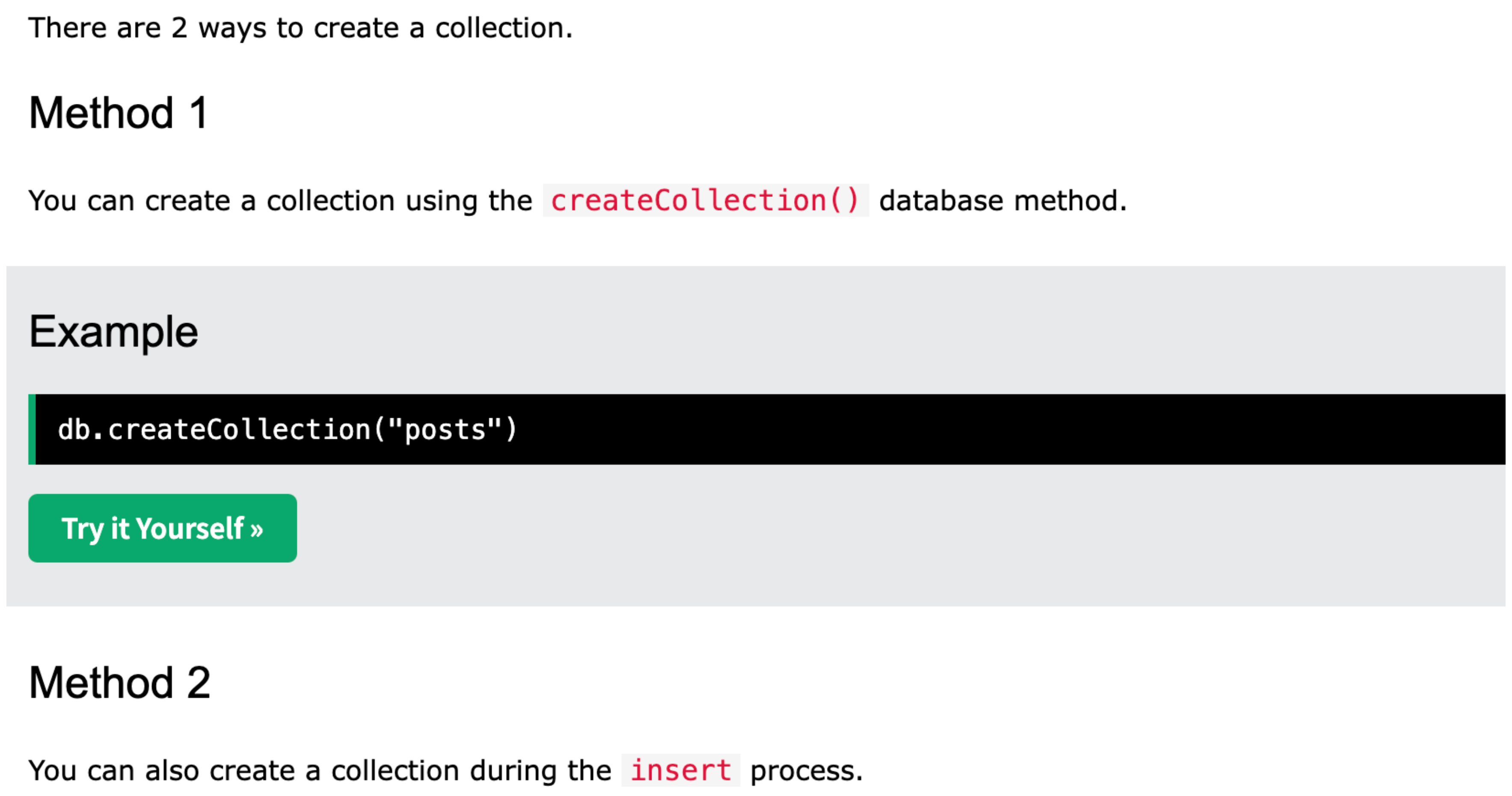
Example

Create a new database called "blog":

```
use blog
```

We are now in the `blog` database.

Create Collection using mongosh



There are 2 ways to create a collection.

Method 1

You can create a collection using the `createCollection()` database method.

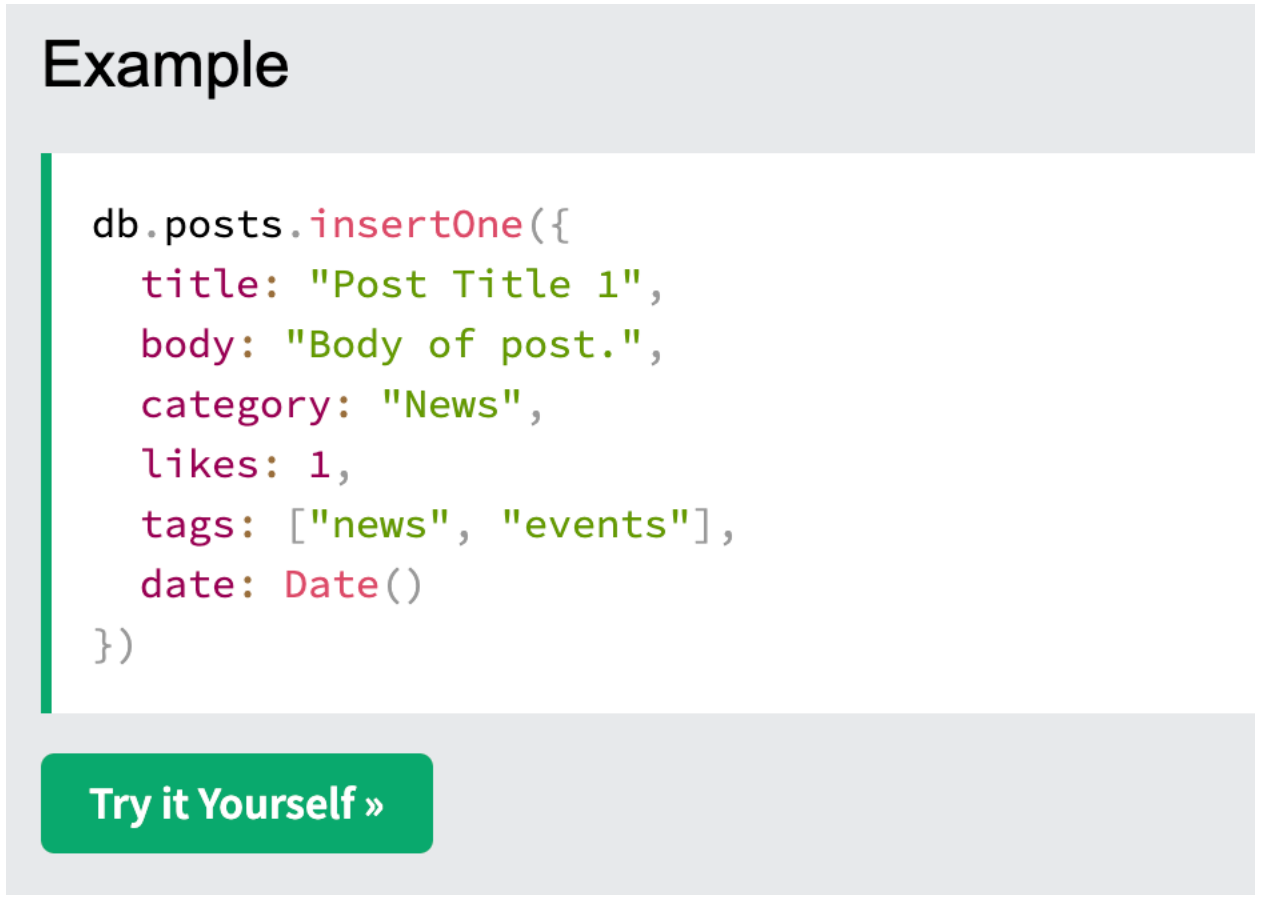
Example

```
db.createCollection("posts")
```

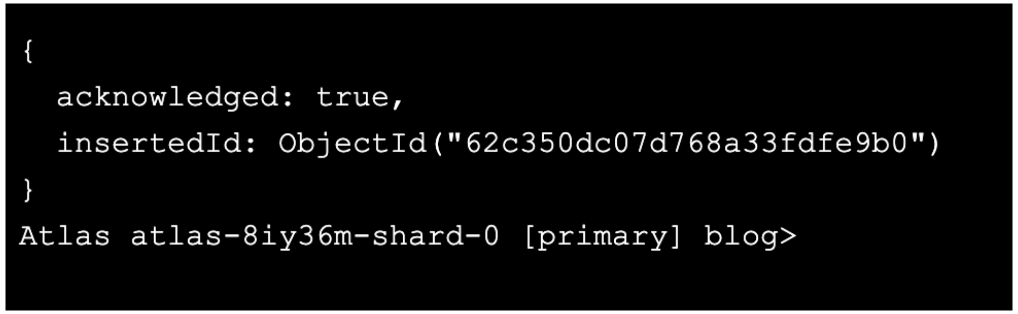
Method 2

You can also create a collection during the `insert` process.

Insert Document (insertOne)



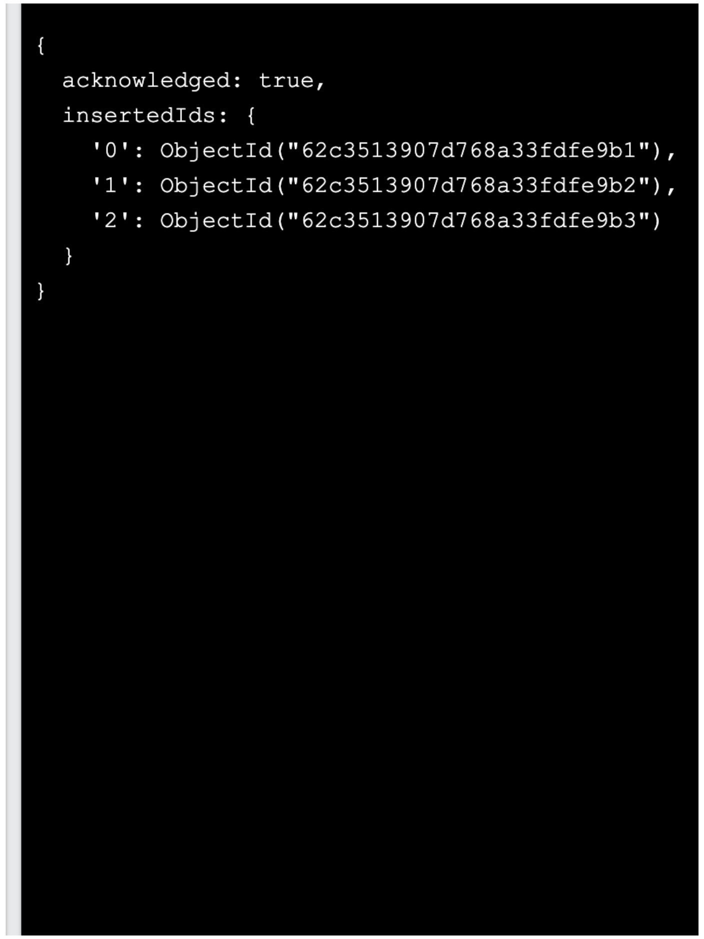
```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```



```
{
  acknowledged: true,
  insertedId: ObjectId("62c350dc07d768a33fdfe9b0")
}
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

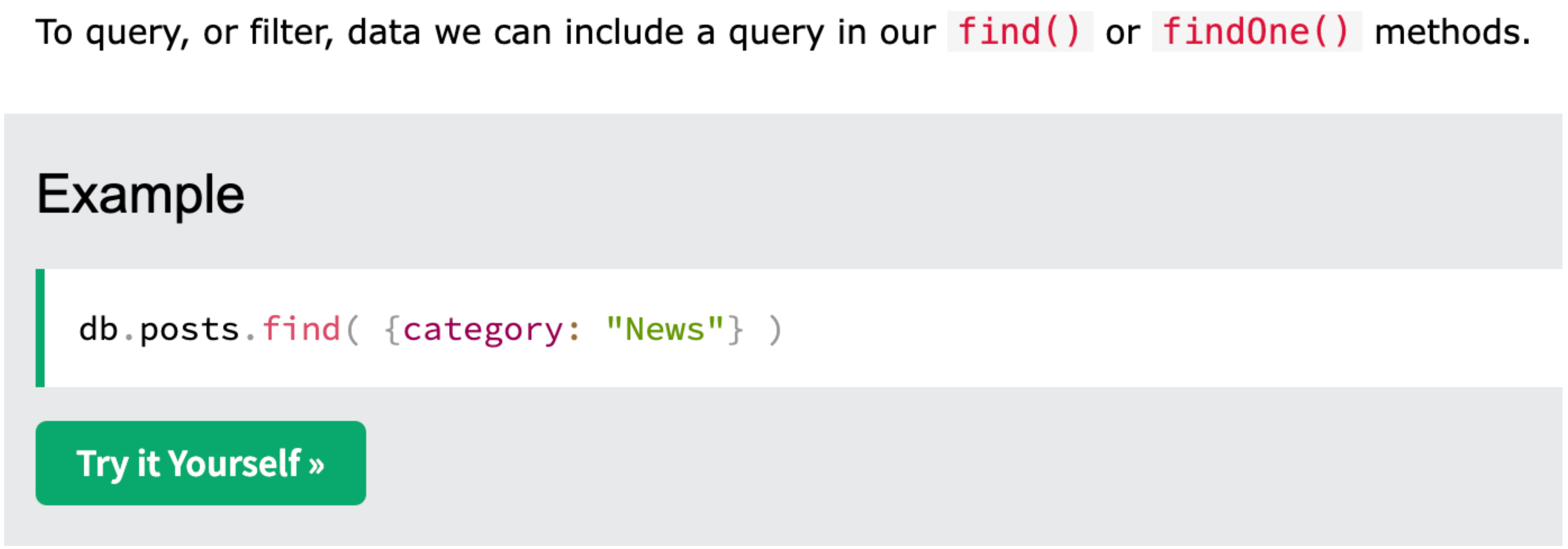
Insert Document (insertMany)

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 4",
    body: "Body of post.",
    category: "Event",
    likes: 4,
    tags: ["news", "events"],
    date: Date()
  }
])
```



```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("62c3513907d768a33fdfe9b1"),
    '1': ObjectId("62c3513907d768a33fdfe9b2"),
    '2': ObjectId("62c3513907d768a33fdfe9b3")
  }
}
```

Querying Data

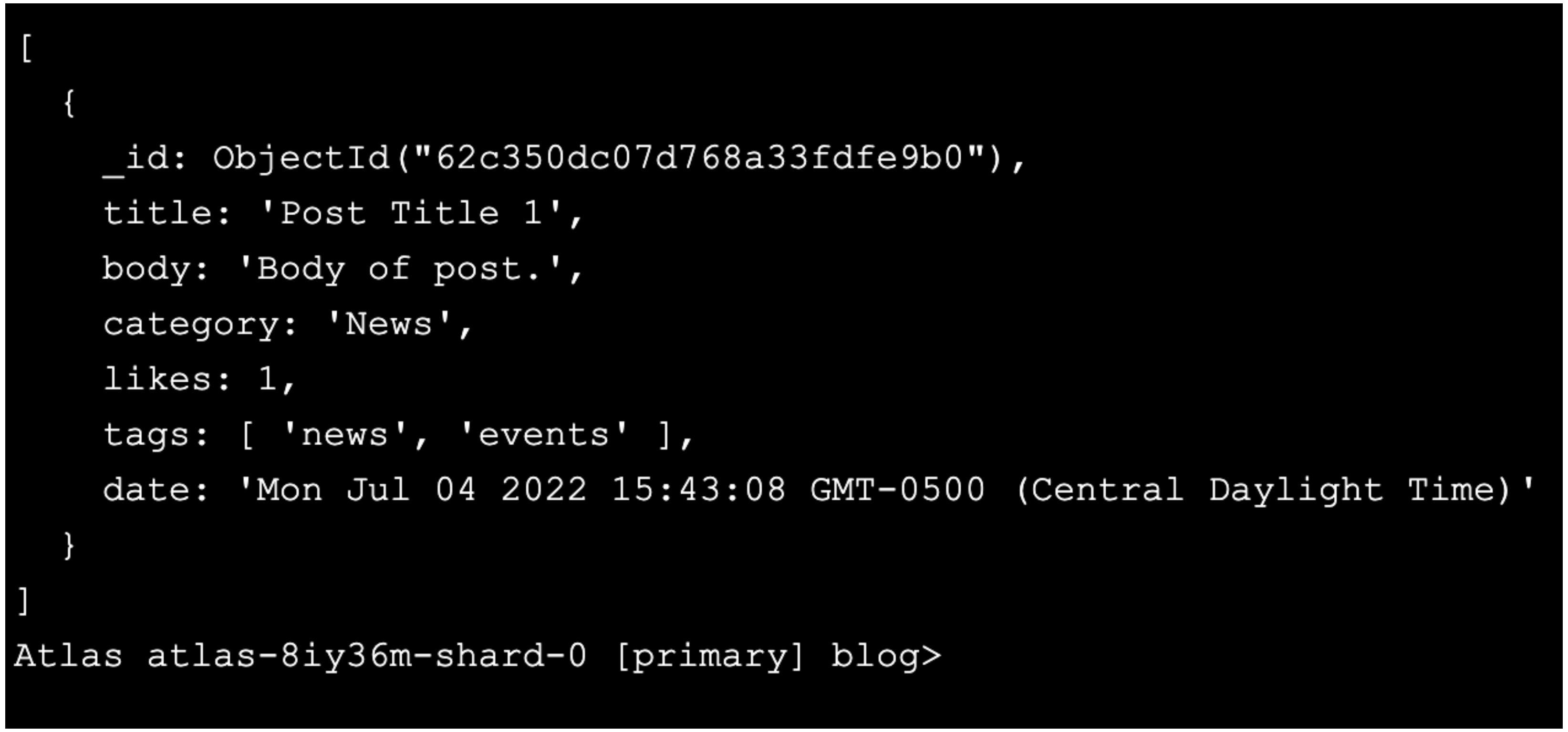


To query, or filter, data we can include a query in our `find()` or `findOne()` methods.

Example

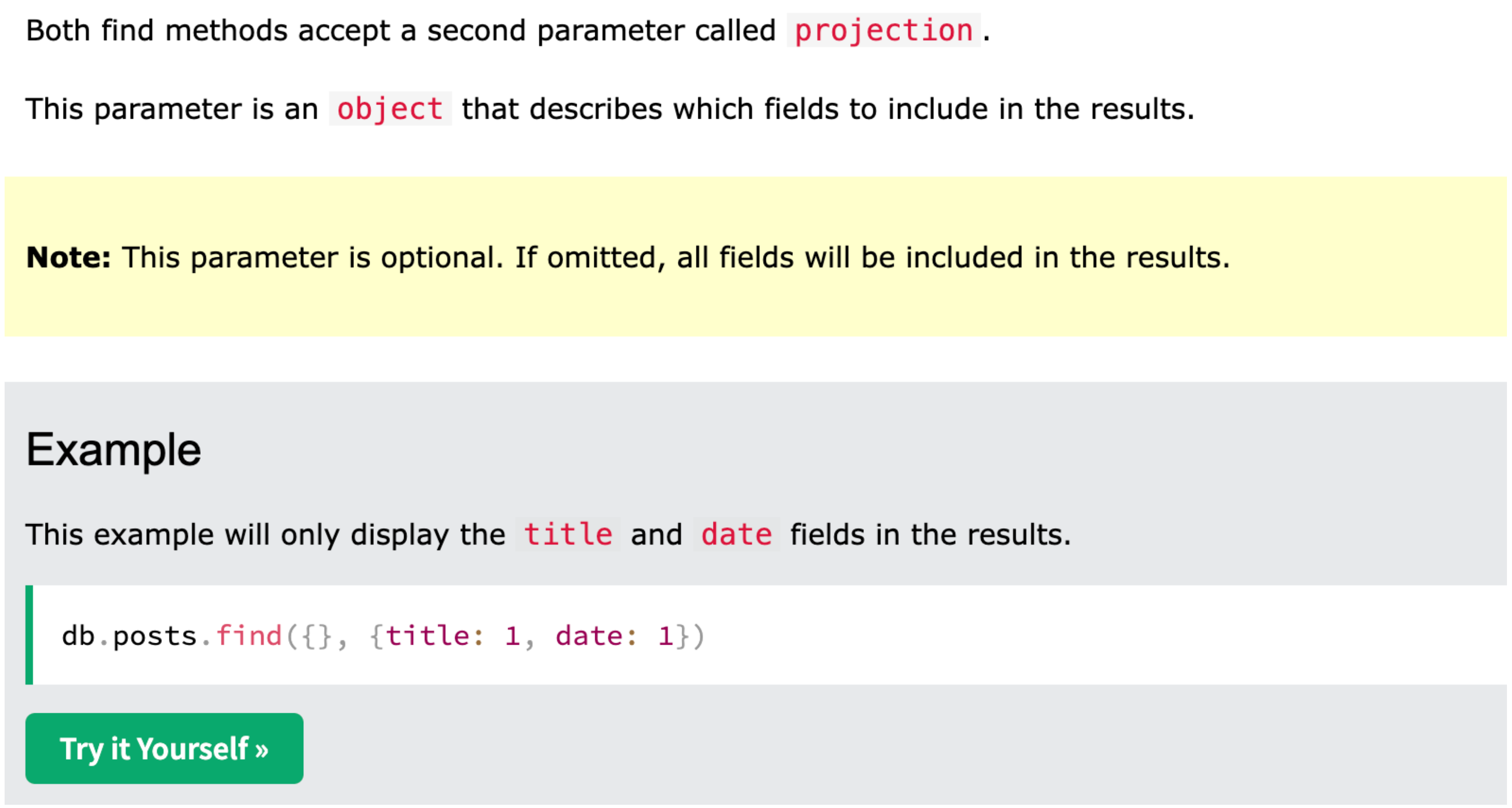
```
db.posts.find( {category: "News"} )
```

Querying Data



```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    body: 'Body of post.',
    category: 'News',
    likes: 1,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  }
]
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

Projection



Both find methods accept a second parameter called `projection`.

This parameter is an `object` that describes which fields to include in the results.

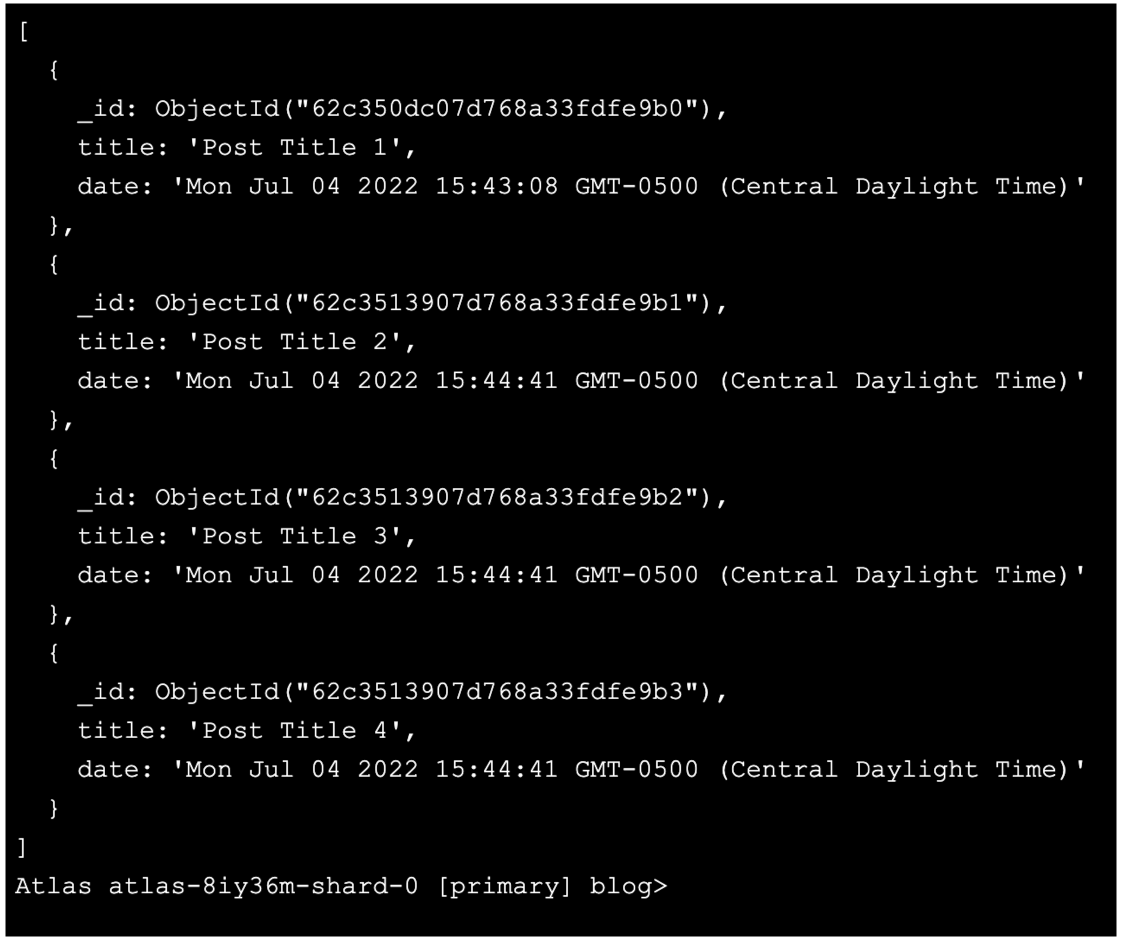
Note: This parameter is optional. If omitted, all fields will be included in the results.

Example

This example will only display the `title` and `date` fields in the results.

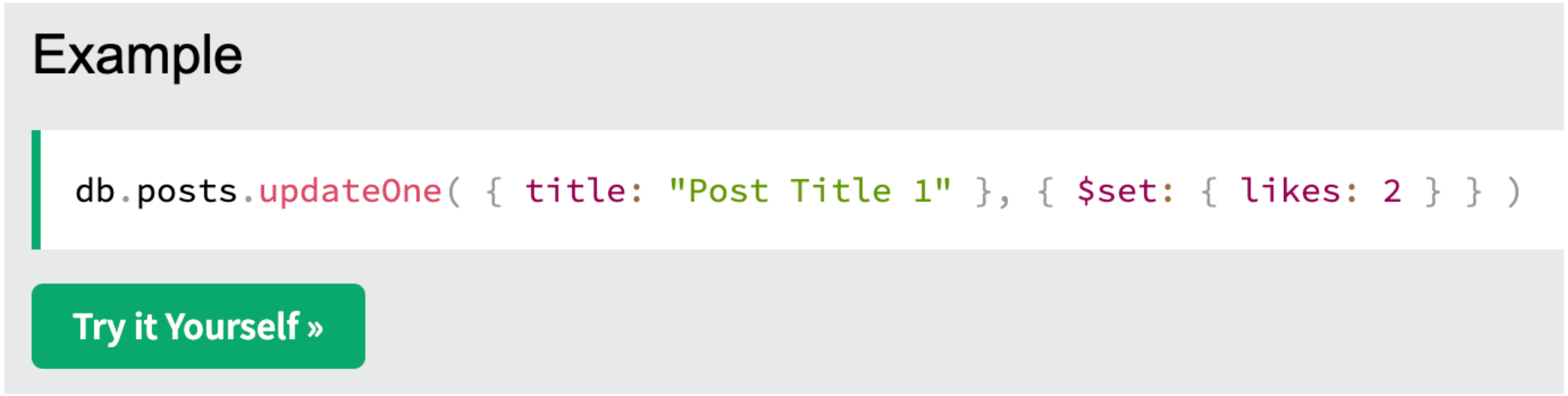
```
db.posts.find({}, {title: 1, date: 1})
```

Projection



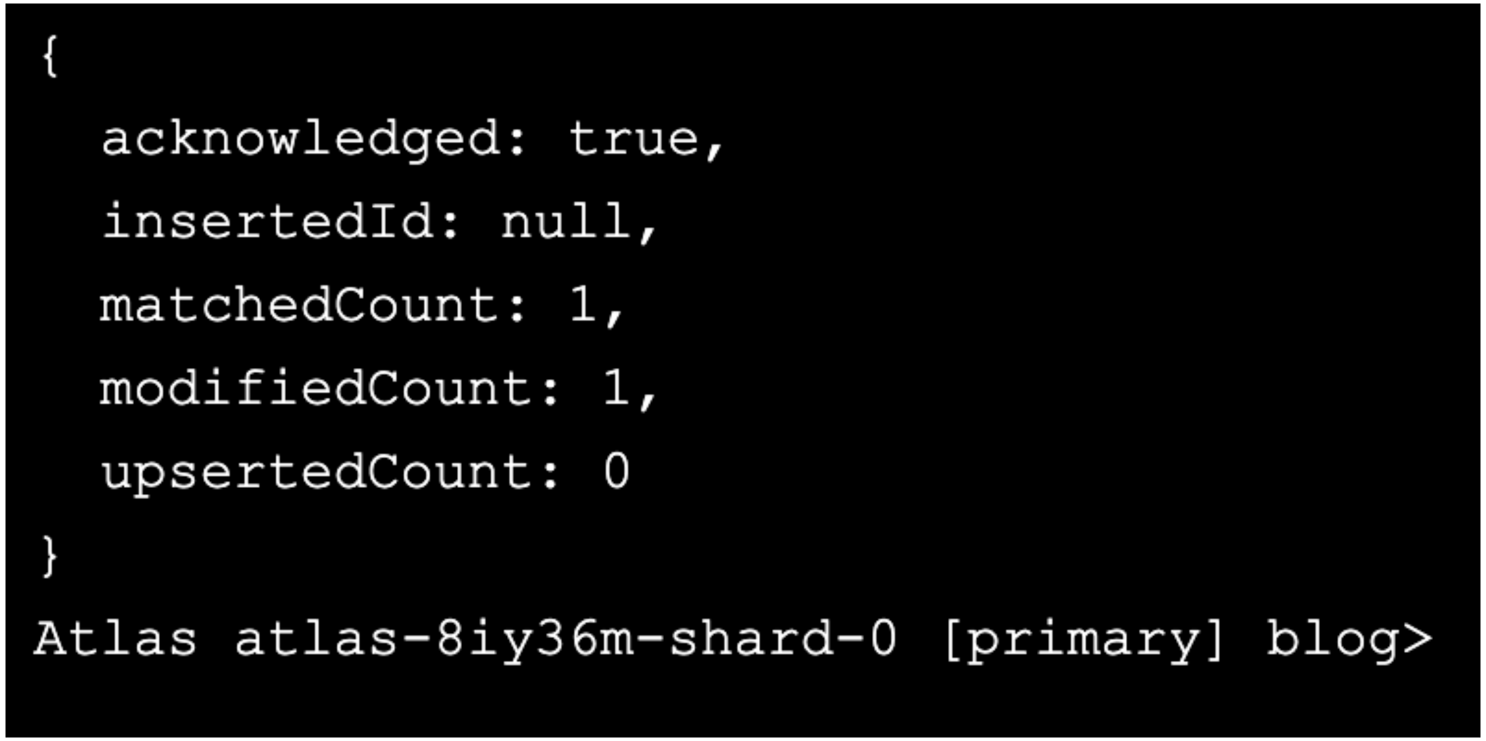
```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  },
  {
    _id: ObjectId("62c3513907d768a33fdfe9b1"),
    title: 'Post Title 2',
    date: 'Mon Jul 04 2022 15:44:41 GMT-0500 (Central Daylight Time)'
  },
  {
    _id: ObjectId("62c3513907d768a33fdfe9b2"),
    title: 'Post Title 3',
    date: 'Mon Jul 04 2022 15:44:41 GMT-0500 (Central Daylight Time)'
  },
  {
    _id: ObjectId("62c3513907d768a33fdfe9b3"),
    title: 'Post Title 4',
    date: 'Mon Jul 04 2022 15:44:41 GMT-0500 (Central Daylight Time)'
  }
]
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

Update Document (updateOne())


Example

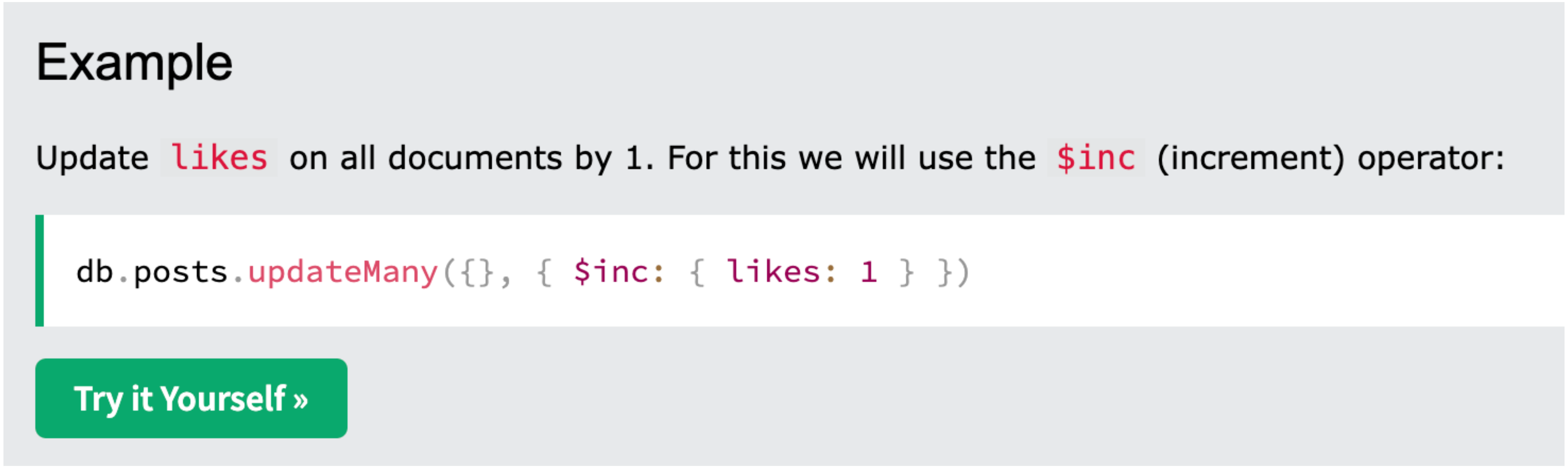
```
db.posts.updateOne( { title: "Post Title 1" }, { $set: { likes: 2 } } )
```

Update Document



```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

Update Document (updateMany())

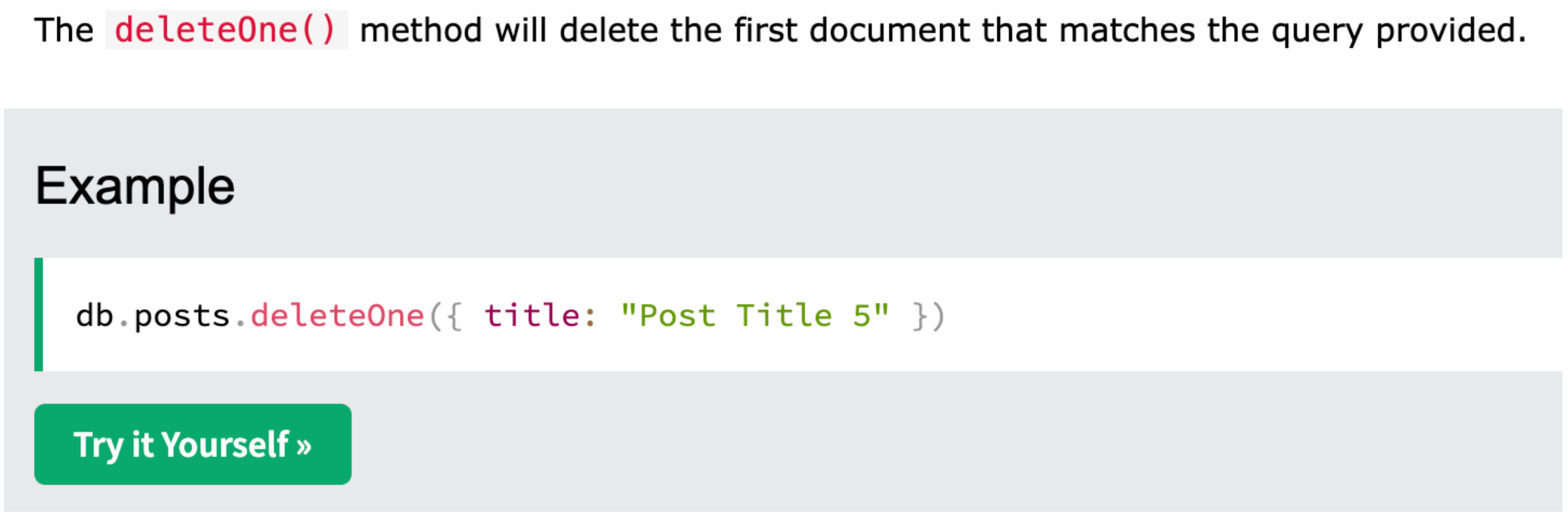


Example

Update `likes` on all documents by 1. For this we will use the `$inc` (increment) operator:

```
db.posts.updateMany({}, { $inc: { likes: 1 } })
```

Delete Document (deleteOne())



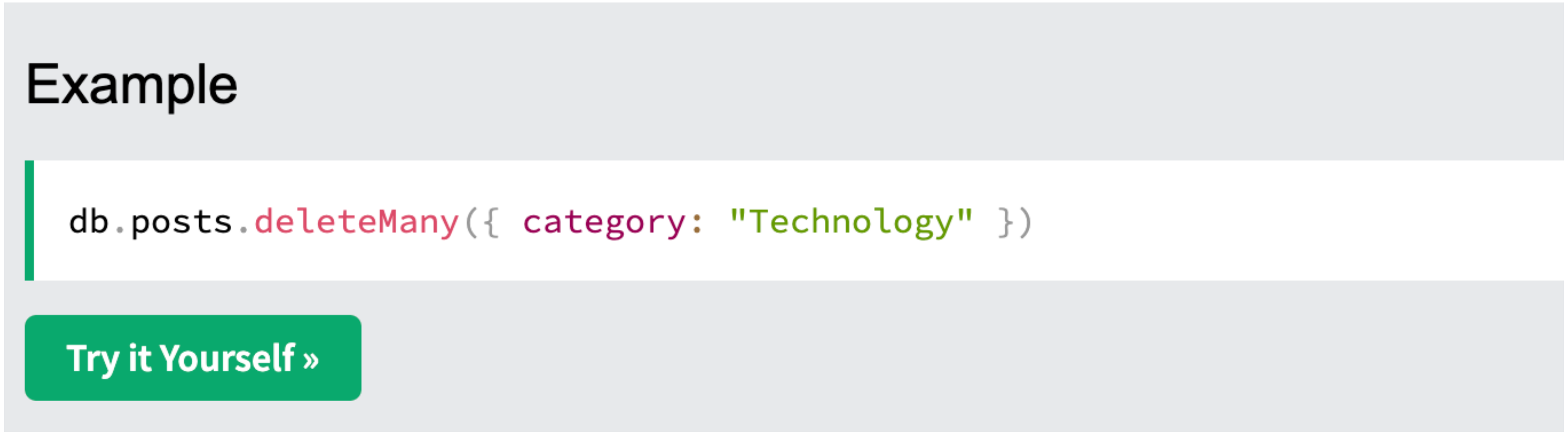
The `deleteOne()` method will delete the first document that matches the query provided.

Example

```
db.posts.deleteOne({ title: "Post Title 5" })
```

Delete Document (deleteMany())

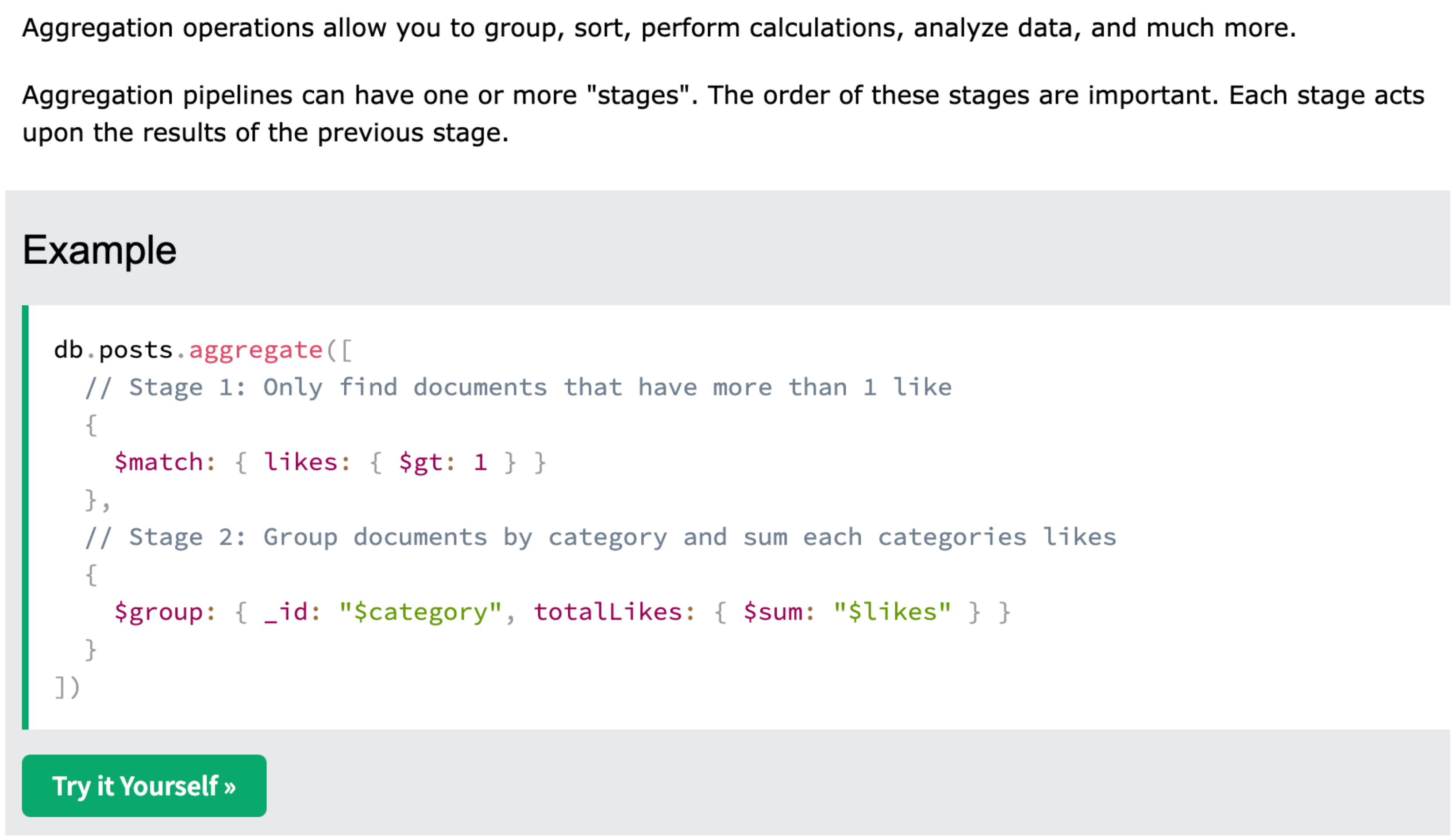
The `deleteMany()` method will delete all documents that match the query provided.



Example

```
db.posts.deleteMany({ category: "Technology" })
```

Aggregation Pipelines



Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more.

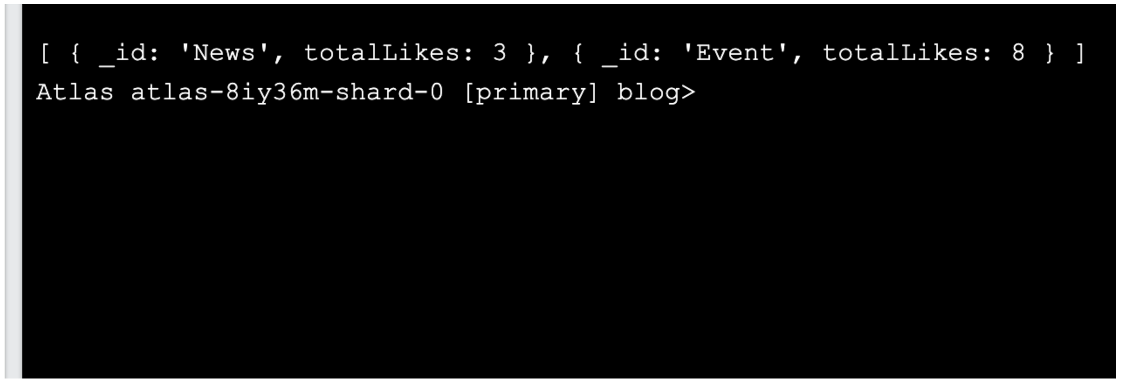
Aggregation pipelines can have one or more "stages". The order of these stages are important. Each stage acts upon the results of the previous stage.

Example

```
db.posts.aggregate([
  // Stage 1: Only find documents that have more than 1 like
  {
    $match: { likes: { $gt: 1 } }
  },
  // Stage 2: Group documents by category and sum each categories likes
  {
    $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
  }
])
```

Aggregation Pipelines

```
db.posts.aggregate([
  {
    $match: { likes: { $gt: 1 } }
  },
  {
    $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
  }
])
```



```
[ { _id: 'News', totalLikes: 3 }, { _id: 'Event', totalLikes: 8 } ]
Atlas atlas-8iy36m-shard-0 [primary] blog>
```