MongoDB Multi-Document ACID Transactions

June 2018

Table of Contents

Introduction

MongoDB 4.0 adds support for multi-document ACID transactions. This makes MongoDB the only open source database to combine the ACID guarantees of traditional relational databases, the speed, flexibility, and power of the document model, with the intelligent distributed systems design to scale-out and place data where you need it.

While existing MongoDB's atomic single-document operations already provide transaction semantics that meet the data integrity needs of the majority of applications, the addition of multi-document ACID transactions makes it easier than ever for developers to address the full spectrum of use cases with MongoDB. Through snapshot isolation, transactions provide a consistent view of data and enforce all-or-nothing execution to maintain data integrity. For developers with a history of transactions in relational databases, MongoDB's multi-document transactions are very familiar, making it straightforward to add them to any application that requires them.

In MongoDB 4.0, transactions work across a replica set, and MongoDB 4.2 will extend support to transactions across a sharded deployment* (see the Safe Harbor statement at the end of this whitepaper). Our path to transactions represents a multi-year engineering effort, beginning back in early 2015 with the groundwork laid in almost every part of the server and the drivers. We are feature complete in bringing multi-document transactions to a replica set, and 90% done on implementing the remaining features needed to deliver transactions across a sharded cluster.

So what does MongoDB 4.0 provide to make it the best way for you to work with your data?

- Open source: **check**
- Flexible, rich data modeling with schema validation: **check**
- Fully expressive joins, faceted search, graphs queries, powerful aggregations, views: **check**
- Rich ecosystem of drivers idiomatic to a given programming language: **check**
- Native horizontal scale-out with sophisticated data routing and localality controls: **check**
- Multi-node durability with tunable semantics: **check**
- Analytics and BI-ready: **check**

- Encryption everywhere and enterprise-grade security integration: **check**
- Mature management tools for ops automation, wherever your infrastructure run: **check**
- Database as a service in every major public cloud (AWS, Azure, GCP): **check**

Add now, multi-document ACID transactions: **check**.

In this whitepaper, we will explore why MongoDB had added multi-document ACID transactions, their design goals and implementation, best practices for developers, and the engineering investments made over the past 3+ years to lay the foundations for them. You can get started with MongoDB 4.0 now by spinning up your own fully managed, on-demand MongoDB Atlas cluster, or downloading it to run on your own infrastructure.

Why Multi-Document ACID Transactions?

Since its first release in 2009, MongoDB has continuously innovated around a new approach to database design, freeing developers from the constraints of legacy relational databases. A design founded on rich, natural, and flexible documents accessed by idiomatic programming language APIs, enabling developers to build apps 3-5x faster. And a distributed systems architecture to handle more data, place it where users need it, and maintain always-on availability. This approach has enabled developers to create powerful and sophisticated applications in all industries, across a tremendously wide range of use cases.

Figure 1: Organizations innovating with MongoDB

With subdocuments and arrays, documents allow related data to be modeled in a single, rich and natural data structure, rather than spread across separate, related tables composed of flat rows and columns. As a result, MongoDB's existing single document atomicity guarantees can meet the data integrity needs of most applications. In fact, when leveraging the richness and power of the document model, we estimate 80%-90% of applications don't need multi-document transactions at all.

However some developers and DBAs have been conditioned by 40 years of relational data modeling to assume multi-record transactions are a requirement for any database, irrespective of the data model they are built upon. Some are concerned that while multi-record transactions aren't needed by their apps today, they might be in the future. And for some workloads, support for ACID transactions across multiple records is required.

As a result, the addition of multi-document transactions makes it easier than ever for developers to address a complete range of use cases on MongoDB. For some, simply knowing that they are available will assure them that they can evolve their application as needed, and the database will support them.

> *"ACID transactions are a key capability for business critical transactional systems, specifically around commerce processing. No other database has both the power of NoSQL and cross-collection ACID transaction support. This combination will make it easy for developers to write mission critical applications leveraging the power of MongoDB."*
>
> **Dharmesh Panchmatia, Director of E-commerce, Cisco Systems**

Data Models and Transactions

Before looking at multi-document transactions in MongoDB, we want to explore why the data model used by a database impacts the scope of a transaction.

Relational Data Model

Relational databases model an entity's data across multiple records and parent-child tables, and so a transaction needs to be scoped to span those records and tables. The example in Figure 2 shows a contact in our customer

database, modeled in a relational schema. Data is normalized across multiple tables: customer, address, city, country, phone number, topics and associated interests.

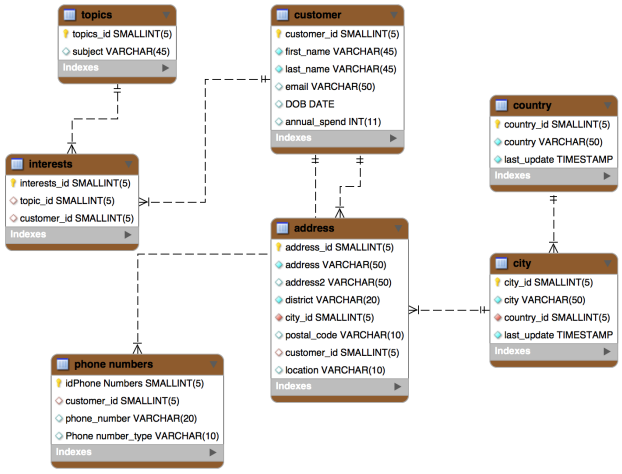

Figure 2: Customer data modeled across separate tables in a relational database

In the event of the customer data changing in any way, for example if our contact moves to a new job, then multiple tables will need to be updated in an "all-or-nothing" transaction that has to touch multiple tables, as illustrated in Figure 3.

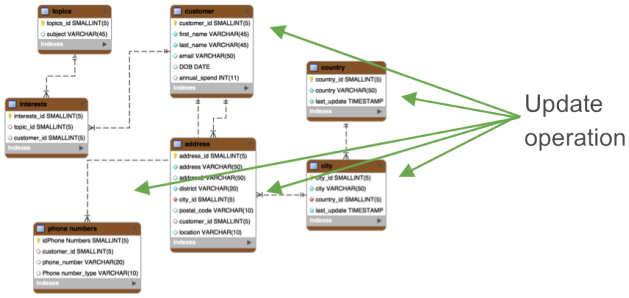

Figure 3: Updating customer data in a relational database

Document Data Model

Document databases are different. Rather than break related data apart and spread it across multiple parent-child tables, documents can store related data together in a rich, typed, hierarchical structure, including subdocuments and arrays, as illustrated in Figure 4.

Figure 4: Customer data modeled in a single, rich document structure

MongoDB provides existing transactional properties scoped to the level of a document. As shown in Figure 5, one or more fields may be written in a single operation, with updates to multiple subdocuments and elements of any array, including nested arrays. The guarantees provided by MongoDB ensure complete isolation as a document is updated; any errors cause the operation to roll back so that clients receive a consistent view of the document. With this design, application owners get the same data integrity guarantees as those provided by legacy relational databases.

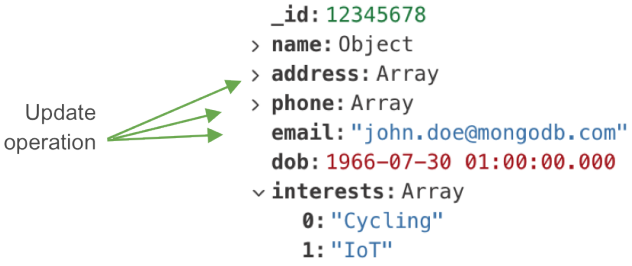

Figure 5: Updating customer data in a document database

Where are Multi-Document Transactions Useful?

There are use cases where transactional ACID guarantees need to be applied to a set of operations that span multiple documents, most commonly with apps that deal with the exchange of value between different parties and require "all-or-nothing" execution. Back office "System of Record" or "Line of Business" (LoB) applications are the typical class of workload where multi-document transactions are useful. Examples include:

- Moving funds between bank accounts, payment processing systems, or trading platforms – for instance where new trades are added to a tick store, while simultaneously updating the risk system and market data dashboards.
- Transferring ownership of goods and services through supply chains and booking systems – for example, inserting a new order in the orders collection and decrementing available inventory in the inventories collection.
- Billing systems – for example adding a new Call Detail Record and then updating the monthly call plan when a cell phone subscriber completes a call.

MongoDB already serves these use cases today, and the introduction of multi-document transactions makes it easier as the database automatically handles multi-document transactions for you. Before their availability, the developer would need to programmatically implement transaction controls in their application. To ensure data integrity, they would need to ensure that all stages of the operation succeed before committing updates to the database, and if not, roll back any changes. This adds complexity that slows down the rate of application development. MongoDB customers in the financial services industry have reported they were able to cut 1,000+ lines of code from their apps by using multi-document transactions.

In addition, implementing client side transactions can impose performance overhead on the application. For example, after moving from its existing client-side transactional logic to multi-document transactions, a global enterprise data management and integration ISV experienced improved MongoDB performance in its Master Data Management solution: throughput increased by 90%, and latency was reduced by over 60% for transactions that performed six updates across two collections. Beyond client side code, a major factor in the performance gains came from write guarantees. For each individual write operation in the transaction, the app previously had to wait for acknowledgement from the majority write concern that the operation had been propagated across a majority of replicas. Only once this acknowledgement was received could it then progress to the next write in the transaction. With multi-document transactions, the app only has to wait for the majority acknowledgement when it comes to commit the transaction.

Multi-Document ACID Transactions in MongoDB

Transactions in MongoDB feel just like transactions developers are used to in relational databases. They are multi-statement, with similar syntax, making them familiar to anyone with prior transaction experience.

The Python code snippet below shows a sample of the transactions API.

```
with client.start_session() as s:
       s.start_transaction()
       collection_one.insert_one(doc_one, session=s)
       collection_two.insert_one(doc_two, session=s)
       s.commit_transaction()
```

The following snippet shows the transactions API for Java.

```
try (ClientSession clientSession
            = client.startSession()) {
         clientSession.startTransaction();
         collection.insertOne(clientSession, docOne);
         collection.insertOne(clientSession, docTwo);
         clientSession.commitTransaction();
    }
```

As these examples show, transactions use regular MongoDB query language syntax, and are implemented consistently whether the transaction is executed across documents and collections in a replica set, and with MongoDB 4.2, across a sharded cluster*.

> *"We're excited to see MongoDB offer dedicated support for ACID transactions in their data platform and that our collaboration is manifest in the Lovelace release of Spring Data MongoDB. It ships with the well known Spring annotation-driven, synchronous transaction support using the MongoTransactionManager but also bits for reactive transactions built on top of MongoDB's ReactiveStreams driver and Project Reactor datatypes exposed via the ReactiveMongoTemplate."*

Pieter Humphrey - Spring Product Marketing Lead, Pivotal

The transaction block code snippets below compare the MongoDB syntax with that used by MySQL. It shows how multi-document transactions feel familiar to anyone who has used traditional relational databases in the past.

MySQL

```
db.start_transaction()
    cursor.execute(orderInsert, orderData)
    cursor.execute(stockUpdate, stockData)
db.commit()
```

MongoDB

```
s.start_transaction()
    orders.insert_one(order, session=s)
    stock.update_one(item, stockUpdate,
        session=s)
s.commit_transaction()
```

Through snapshot isolation, transactions provide a consistent view of data, and enforce all-or-nothing execution to maintain data integrity. Transactions can apply to operations against multiple documents contained in one, or in many, collections and databases. The changes to MongoDB that enable multi-document transactions do not impact performance for workloads that don't require them.

During its execution, a transaction is able to read its own uncommitted writes, but none of uncommitted writes will be seen by other operations outside of the transaction. Uncommitted writes are not replicated to secondary nodes until the transaction is committed to the database. Once the transaction has been committed, it is replicated and applied atomically to all secondary replicas.

Taking advantage of the transactions infrastructure introduced in MongoDB 4.0, the new snapshot read concern ensures queries and aggregations executed within a read-only transaction will operate against a single, isolated snapshot on the primary replica. As a result, a consistent view of the data is returned to the client, irrespective of whether that data is being simultaneously modified by concurrent operations. Snapshot reads are especially useful for operations that return data in batches with the getMore command.

Transactions Best Practices

As noted earlier, MongoDB's existing document atomicity guarantees will meet 80-90% of an application's transactional needs. They remain the recommended way of enforcing your app's data integrity requirements. For those operations that do require multi-document transactions, there are several best practices that developers should observe.

Creating long running transactions, or attempting to perform an excessive number of operations in a single ACID transaction can result in high pressure on WiredTiger's cache. This is because the cache must maintain state for all subsequent writes since the oldest snapshot was created. As a transaction always uses the same snapshot while it is running, new writes accumulate in the cache throughout the duration of the transaction. These writes cannot be flushed until transactions currently running on old snapshots commit or abort, at which time the transactions release their locks and WiredTiger can evict the snapshot. To maintain predictable levels of database performance, developers should therefore consider the following:

1. By default, MongoDB will automatically abort any multi-document transaction that runs for more than 60 seconds. Note that if write volumes to the server are low, you have the flexibility to tune your transactions for a longer execution time. To address timeouts, the transaction should be broken into smaller parts that allow execution within the configured time limit. You should also ensure your query patterns are properly optimized with the appropriate index coverage to allow fast data access within the transaction.
2. There are no hard limits to the number of documents that can be read within a transaction. As a best practice, no more than 1,000 documents should be modified within a transaction. For operations that need to modify more than 1,000 documents, developers should break the transaction into separate parts that process documents in batches.
3. In MongoDB 4.0, a transaction is represented in a single oplog entry, therefore must be within the 16MB document size limit. While an update operation only stores the deltas of the update (i.e., what has changed), an insert will store the entire document. As a result, the

combination of oplog descriptions for all statements in the transaction must be less than 16MB. If this limit is exceeded, the transaction will be aborted and fully rolled back. The transaction should therefore be decomposed into a smaller set of operations that can be represented in 16MB or less.

4. When a transaction aborts, an exception is returned to the driver and the transaction is fully rolled back. Developers should add application logic that can catch and retry a transaction that aborts due to temporary exceptions, such as a transient network failure or a primary replica election. With retryable writes, the MongoDB drivers will automatically retry the commit statement of the transaction.
5. DDL operations, like creating an index or dropping a database, block behind active running transactions on the namespace. All transactions that try to newly access the namespace while DDL operations are pending, will not be able to obtain locks, aborting the new transactions.

You can review all best practices in the MongoDB documentation for multi-document transactions.

Transactions and Their Impact to Data Modeling in MongoDB

Adding transactions does not make MongoDB a relational database – many developers have already experienced that the document model is superior to the relational one today.

All best practices relating to MongoDB data modeling continue to apply when using features such as multi-document transactions, or fully expressive JOINs (via the $lookup aggregation pipeline stage). Where practical, all data relating to an entity should be stored in a single, rich document structure. Just moving data structured for relational tables into MongoDB will not allow users to take advantage of MongoDB's natural, fast, and flexible document model, or its distributed systems architecture.

The RDBMS to MongoDB Migration Guide describes the best practices for moving an application from a relational database to MongoDB.

The Path to Transactions

Our path to transactions represents a multi-year engineering effort, beginning over 3 years ago with the integration of the WiredTiger storage engine. We've laid the groundwork in practically every part of the platform – from the storage layer itself to the replication consensus protocol, to the sharding architecture. We've built out fine-grained consistency and durability guarantees, introduced a global logical clock, refactored cluster metadata management, and more. And we've exposed all of these enhancements through APIs that are fully consumable by our drivers. We are feature complete in bringing multi-document transactions to a replica set, and 90% done on implementing the remaining features needed to deliver transactions across a sharded cluster.

Figure 6 presents a timeline of the key engineering projects that have enabled multi-document transactions in MongoDB, with status shown as of June 2018. The key design goal underlying all of these projects is that their implementation does not sacrifice the key benefits of MongoDB – the power of the document model and the advantages of distributed systems, while imposing no performance impact to workloads that don't require multi-document transactions.

One of the hardest parts of the engineering effort has been how to balance two priorities. Firstly, building the stepping stones we needed to get to multi-document transactions in MongoDB 4.0. And secondly, immediately exposing useful features to our users, to make MongoDB the best way to work with their data. Wherever we could, we built components that satisfied both priorities:

- The acquisition of WiredTiger Inc. and integration of its storage engine way back in MongoDB 3.0 brought massive scalability gains through document level concurrency control and compression. And with MVCC support, it also provided the storage layer foundations for multi-document transactions.
- In MongoDB 3.2, the enhanced consensus protocol allowed for faster and more deterministic recovery from the failure of the primary replica set member or network partitioning, along with stricter durability guarantees for writes. These enhancements were immediately useful to

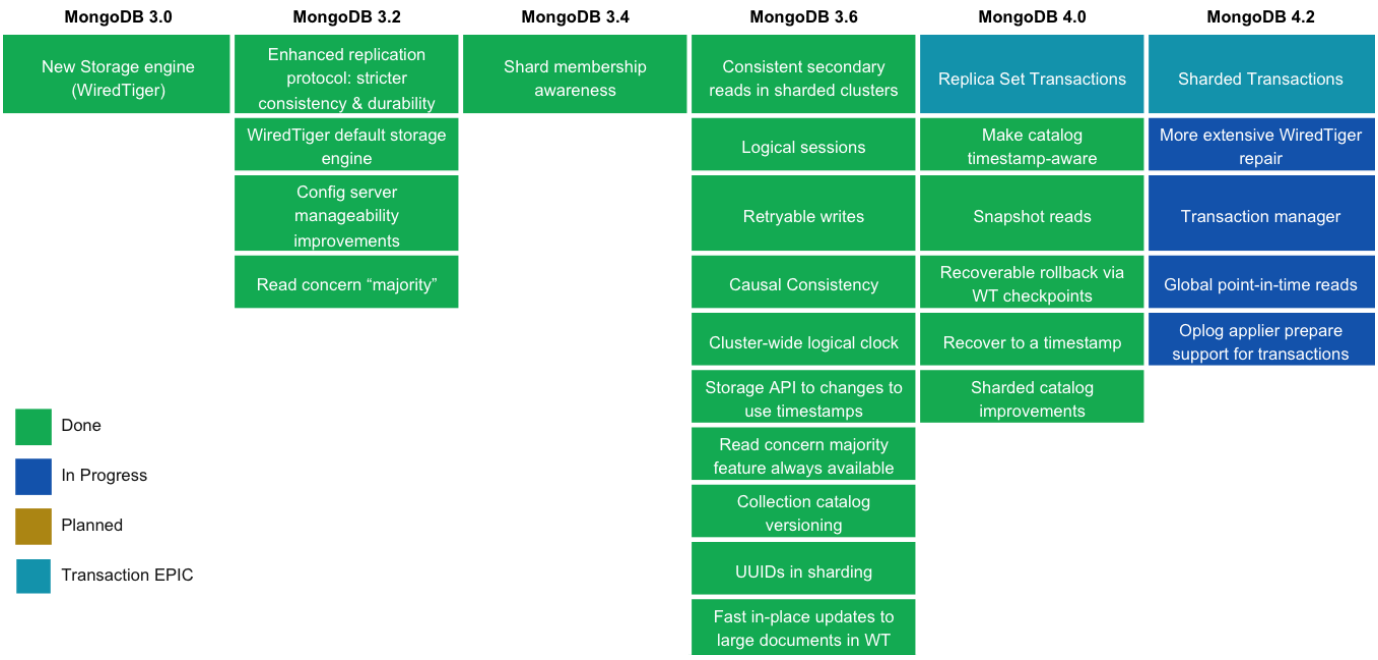

Figure 6: The path to transactions – multi-year engineering investment, delivered across multiple releases

MongoDB users at the time, and they are also essential capabilities for transactions going forward.

- The introduction of the readConcern query option in 3.2 allowed applications to specify the read isolation level on a per operation basis, providing powerful and granular consistency controls for users then, and the isolation levels required for multi-document transactions in MongoDB 4.0 and beyond.

MongoDB 3.6 brought a host of new functionality, much of which was underpinned by the introduction of a global logical clock and storage layer timestamps, based on an implementation of the earlier academic concepts of Lamport clocks and timestamps. This functionality enforces consistent time across every operation in a distributed cluster, enabling multi-document transactions to provide snapshot isolation guarantees. And it also allowed us to expose additional capabilities that further improved developer productivity as soon as MongoDB 3.6 was released:

- Change streams enable developers to build reactive applications that can view, filter, and act on data changes as they occur in the database, in real time. The logical clock and timestamps give change streams resumability – automatically recovering from transient node failures, so consuming apps can just pick up applying changes from the point when the node failure occurred.
- Logical sessions are the foundation for both causal consistency and retryable writes, discussed below. From the perspective of multi-document transactions, their value is the ability to coordinate client and server operations across distributed nodes, managing the execution context for each statement in a transaction.
- Causal consistency, enabled by logical sessions and cluster time, allows developers to maintain the benefits of strong data consistency with "read your own write" guarantees, while taking advantage of the scalability and availability properties of our intelligent distributed data platform.
- Retryable writes simplify the development of applications in the face of elections (or other transient failures), while the server enforces exactly-once processing semantics.

The number of remaining pieces on the roadmap to sharded transactions is small. All of the work needed to bring multi-document transactions to replica sets is done, and we are in progress on the remaining work needed to extend transactions to sharded clusters in the next major MongoDB release.

Conclusion

MongoDB has already established itself as the leading database for modern applications. The document data model is rich, natural, and flexible, with documents accessed by idiomatic drivers, enabling developers to build apps 3-5x faster. Its distributed systems architecture enables you to handle more data, place it where users need it, and maintain always-on availability. MongoDB's existing atomic single-document operations provide transaction semantics that meet the data integrity needs of the majority of applications. The addition of multi-document ACID transactions in MongoDB 4.0 makes it easier than ever for developers to address a complete range of us cases, while for many, simply knowing that they are available will provide critical peace of mind.

Take a look at our multi-document transactions web page where you can hear directly from the MongoDB engineers who have built transactions, review code snippets, and access key resources to get started. You can get started with MongoDB 4.0 now by spinning up your own fully managed, on-demand MongoDB Atlas cluster, or downloading it to run on your own infrastructure.

> *"Transactional guarantees have been a critical feature for relational databases for decades, but have typically been absent from non-relational alternatives, which has meant that users have been forced to choose between transactions and the flexibility and versatility that non-relational databases offer. With its support for multi-document ACID transactions, MongoDB is built for customers that want to have their cake and eat it too."*

Stephen O'Grady, Principal Analyst, Redmonk

*Safe Harbour Statement

This paper contains "forward-looking statements" within the meaning of Section 27A of the Securities Act of 1933, as amended, and Section 21E of the Securities Exchange Act of 1934, as amended. Such forward-looking statements are subject to a number of risks, uncertainties, assumptions and other factors that could cause actual results and the timing of certain events to differ materially from future results expressed or implied by the forward-looking statements. Factors that could cause or contribute to such differences include, but are not limited to, those identified our filings with the Securities and Exchange Commission. You should not rely upon forward-looking statements as predictions of future events. Furthermore, such forward-looking statements speak only as of the date of this presentation.

In particular, the development, release, and timing of any features or functionality described for MongoDB products remains at MongoDB's sole discretion. This information is merely intended to outline our general product direction and it should not be relied on in making a purchasing decision nor is this a commitment, promise or legal obligation to deliver any material, code, or functionality. Except as required by law, we undertake no obligation to update any forward-looking statements to reflect events or circumstances after the date of such statements.

We Can Help

We are the MongoDB experts. Over 6,600 organizations rely on our commercial products. We offer software and services to make your life easier:

MongoDB Enterprise Advanced is the best way to run MongoDB in your data center. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Atlas is a database as a service for MongoDB, letting you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. With the click of a button, you can scale up and down when you need to, with no downtime, full security, and high performance.

MongoDB Stitch is a serverless platform which accelerates application development with simple, secure access to data and services from the client – getting your apps to market faster while reducing operational costs and effort.

MongoDB Mobile (Beta) MongoDB Mobile lets you store data where you need it, from IoT, iOS, and Android mobile devices to your backend – using a single database and query language.

MongoDB Cloud Manager is a cloud-based tool that helps you manage MongoDB on your own infrastructure. With automated provisioning, fine-grained monitoring, and continuous backups, you get a full management suite that reduces operational overhead, while maintaining full control over your databases.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.

Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.com)
MongoDB Enterprise Download (mongodb.com/download)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)
MongoDB Stitch backend as a service (mongodb.com/cloud/stitch)

mongoDB

US 866-237-8815 ▪ INTL +1-650-440-4474 ▪ info@mongodb.com