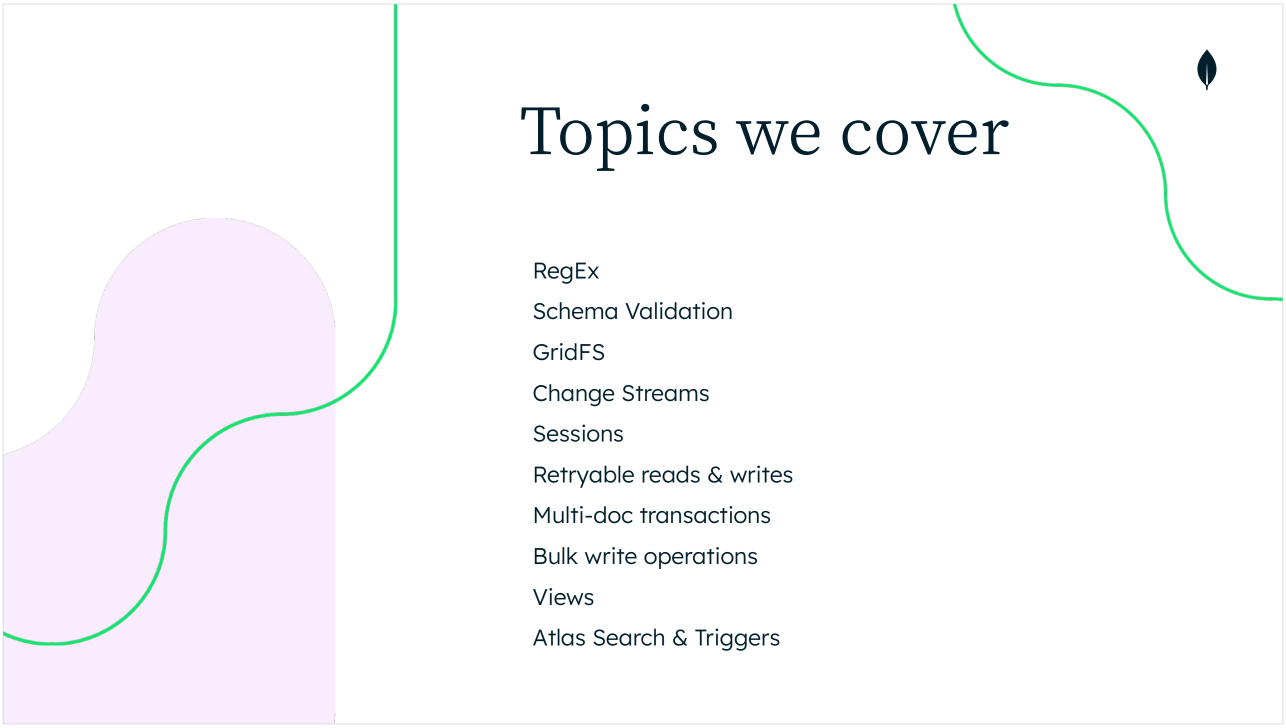
DF300

Beyond Storage

Design Skills and Advanced Features

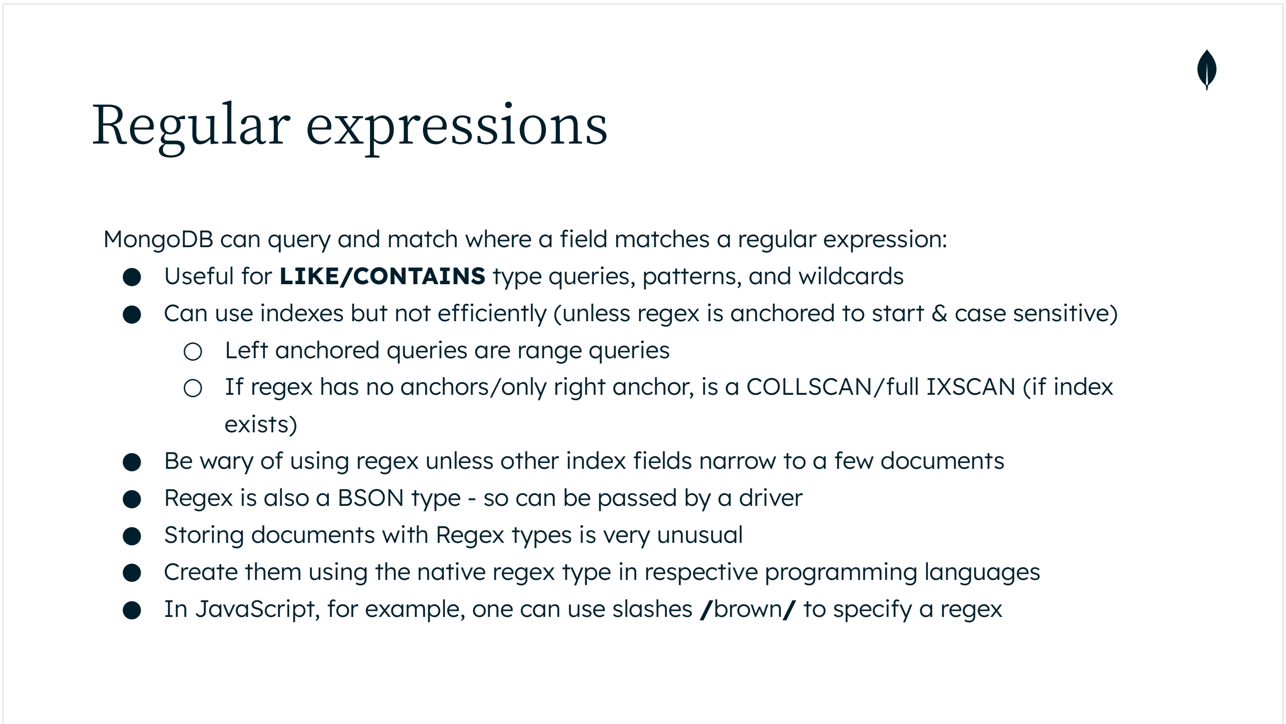
Release: 20230414



Topics we cover

- RegEx
- Schema Validation
- GridFS
- Change Streams
- Sessions
- Retryable reads & writes
- Multi-doc transactions
- Bulk write operations
- Views
- Atlas Search & Triggers

For next set of slides - as a class the instructor will ask you to suggest two ways you might use that capability in a project you are working on

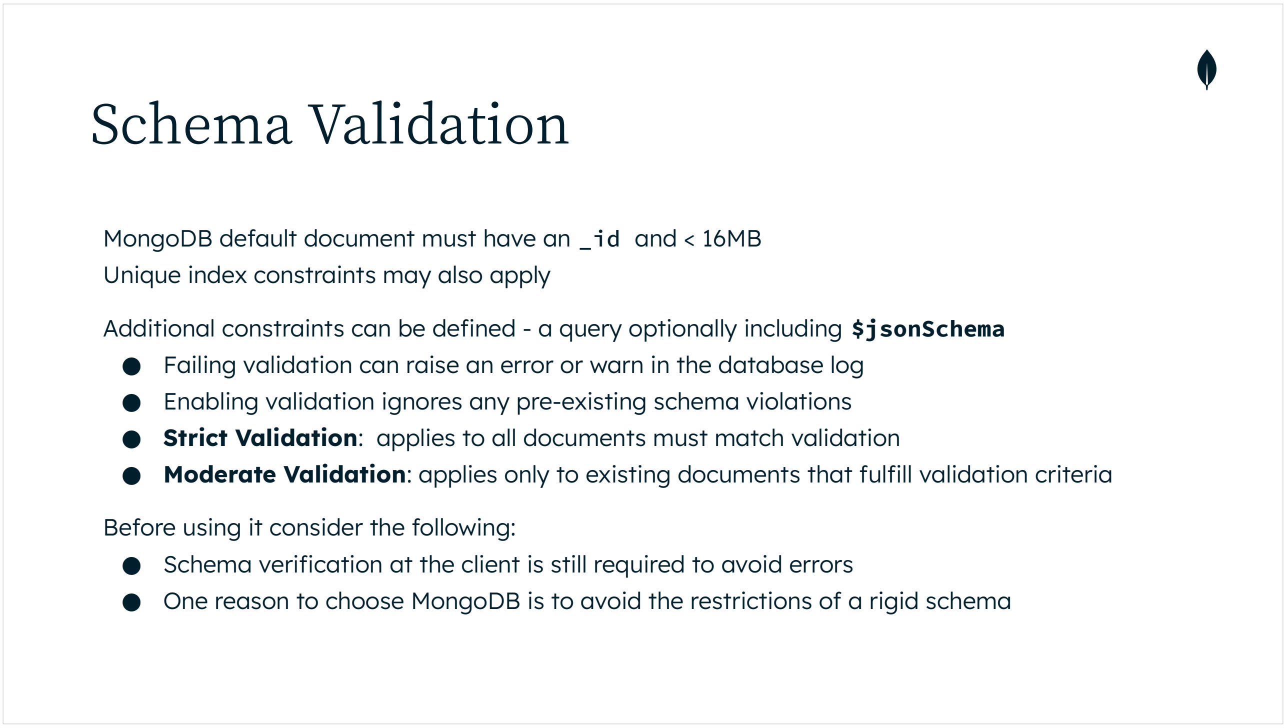


Regular expressions

MongoDB can query and match where a field matches a regular expression:

- Useful for **LIKE/CONTAINS** type queries, patterns, and wildcards
- Can use indexes but not efficiently (unless regex is anchored to start & case sensitive)
  - ○ Left anchored queries are range queries
  - ○ If regex has no anchors/only right anchor, is a COLLSCAN/full IXSCAN (if index exists)
- Be wary of using regex unless other index fields narrow to a few documents
- Regex is also a BSON type - so can be passed by a driver
- Storing documents with Regex types is very unusual
- Create them using the native regex type in respective programming languages
- In JavaScript, for example, one can use slashes **/**brown**/** to specify a regex

- Let's start with some CRUD functionality.
- BSON has a regex data type - this is more to allow you to send a regex query to the server than to store regexes - although you can.
- This allows us to do wildcards, contains, like, and other fuzzy matching.
- However, anything beyond case sensitive won't be indexed and non left-anchor (starts-with) queries might not be efficient- so use regex sparingly; it's not a panacea.
- When using a regex query it is easy to forget that `{name: /^smith/}` is actually `{ name: { $gte : "smith", $lt: "smiti"}}` and therefore compound index ordering for ranges applies.
- Regex queries can become range queries when anchored to the left with ^ (e.g. `/^brown/`) and can use an index efficiently
- A non anchored, non case sensitive might still do a full index SCAN rather than a collection scan.
- Case insensitive queries should use collation (not regular expression) and case insensitive regex will **never** use an index.



Schema Validation

MongoDB default document must have an `_id` and < 16MB
Unique index constraints may also apply

Additional constraints can be defined - a query optionally including **`$jsonSchema`**

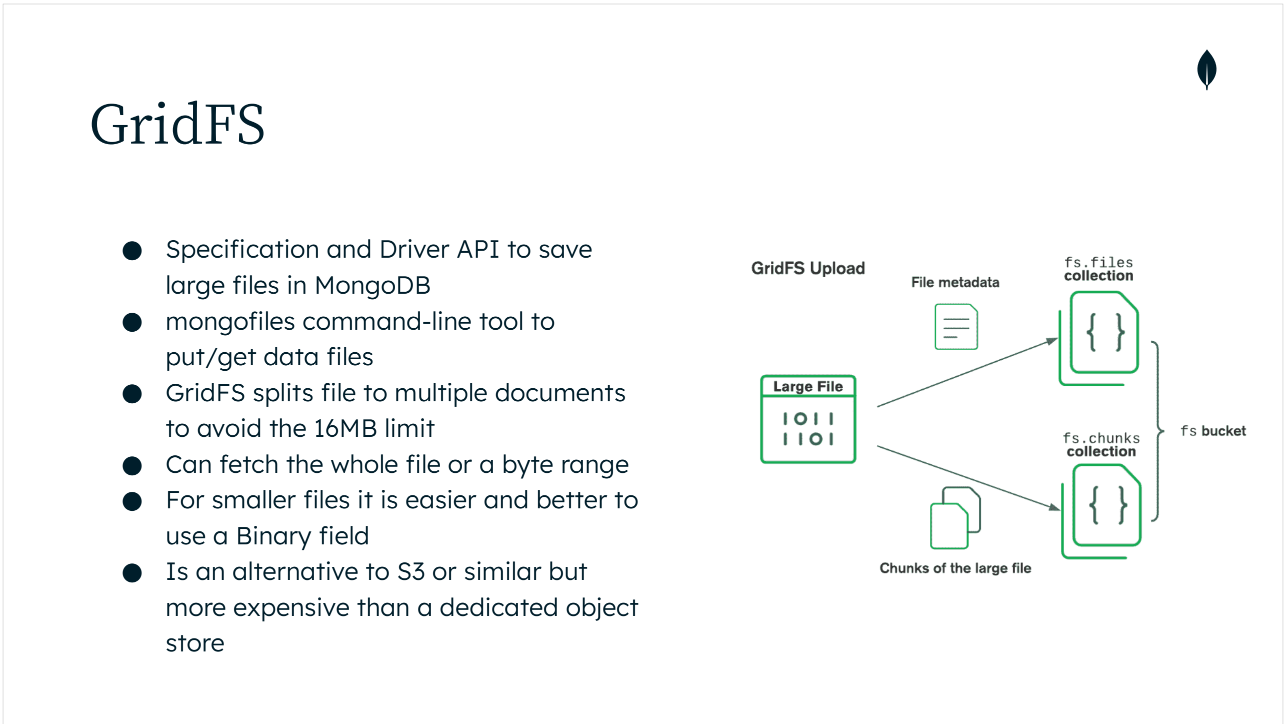
- Failing validation can raise an error or warn in the database log
- Enabling validation ignores any pre-existing schema violations
- **Strict Validation**: applies to all documents must match validation
- **Moderate Validation**: applies only to existing documents that fulfill validation criteria

Before using it consider the following:

- Schema verification at the client is still required to avoid errors
- One reason to choose MongoDB is to avoid the restrictions of a rigid schema

---

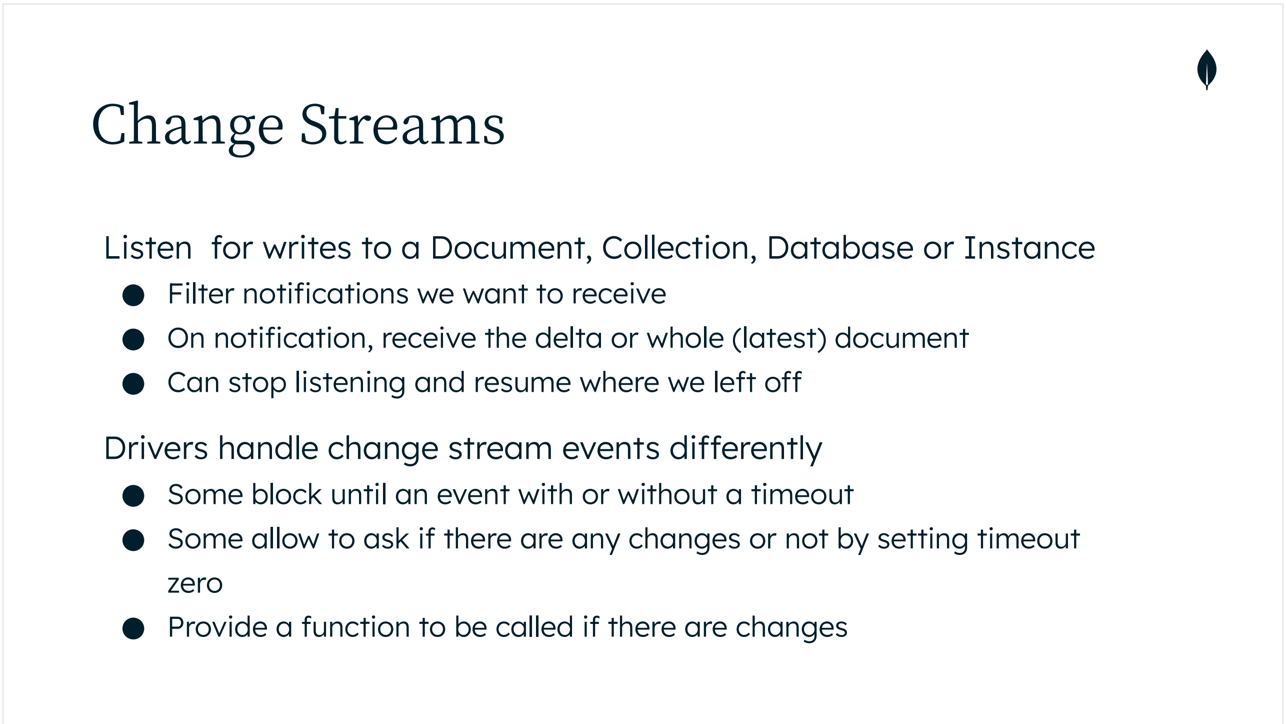
- The MongoDB server applies very few constraints on documents by default, _id is unique and not an array and documents have a max size.
- You can ask it to apply more constraints if you want.
- These can be hard constraints (fail a write) or just log violations in the database log
- It can cope with retroactively applying a constraint where older data violates it.
- Schema validation is not generally recommended as client end enforcement is needed anyway
- MongoDB can allow writes to invalid documents if they already exist and are invalid.
- Sometimes used where there is no centralized management of code/database and many teams have the write permissions



GridFS

- Specification and Driver API to save large files in MongoDB
- mongofiles command-line tool to put/get data files
- GridFS splits file to multiple documents to avoid the 16MB limit
- Can fetch the whole file or a byte range
- For smaller files it is easier and better to use a Binary field
- Is an alternative to S3 or similar but more expensive than a dedicated object store

- GridFS is a solution to storing large binary files in MongoDB.
- It's debatable if this is a good thing to do. However, some people want to - it does give redundant, HA file storage.
- Files are split into chunks in one collection, and a metadata collection stores info on the filename, size, checksum, etc.
- Chunk size is configurable default is 256KB but that's too small usually better to use 1MB



Change Streams

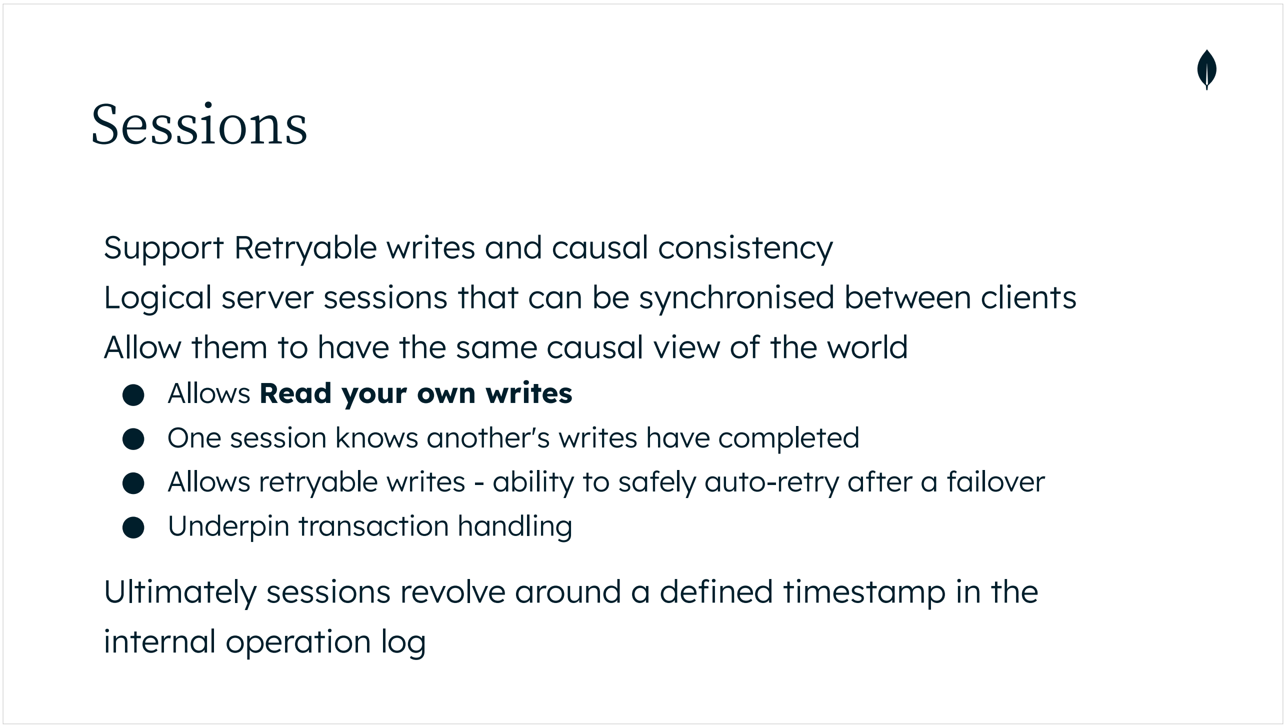
Listen for writes to a Document, Collection, Database or Instance

- Filter notifications we want to receive
- On notification, receive the delta or whole (latest) document
- Can stop listening and resume where we left off

Drivers handle change stream events differently

- Some block until an event with or without a timeout
- Some allow to ask if there are any changes or not by setting timeout zero
- Provide a function to be called if there are changes

- Change streams are a really powerful feature to allow you to build reactive applications - you don't need to poll for changes. You can listen for them.
- If we want the whole document, it's the latest majority committed versions - not always what it was right after that change!
- Listening for them needs threads or some kind of asynchronous language though.
- This used to be done by watching the transaction log (oplog). However, there were many edge cases.
- Change streams only notify on data changes that have persisted to a majority of data-bearing members in the replica set.



Sessions

Support Retryable writes and causal consistency

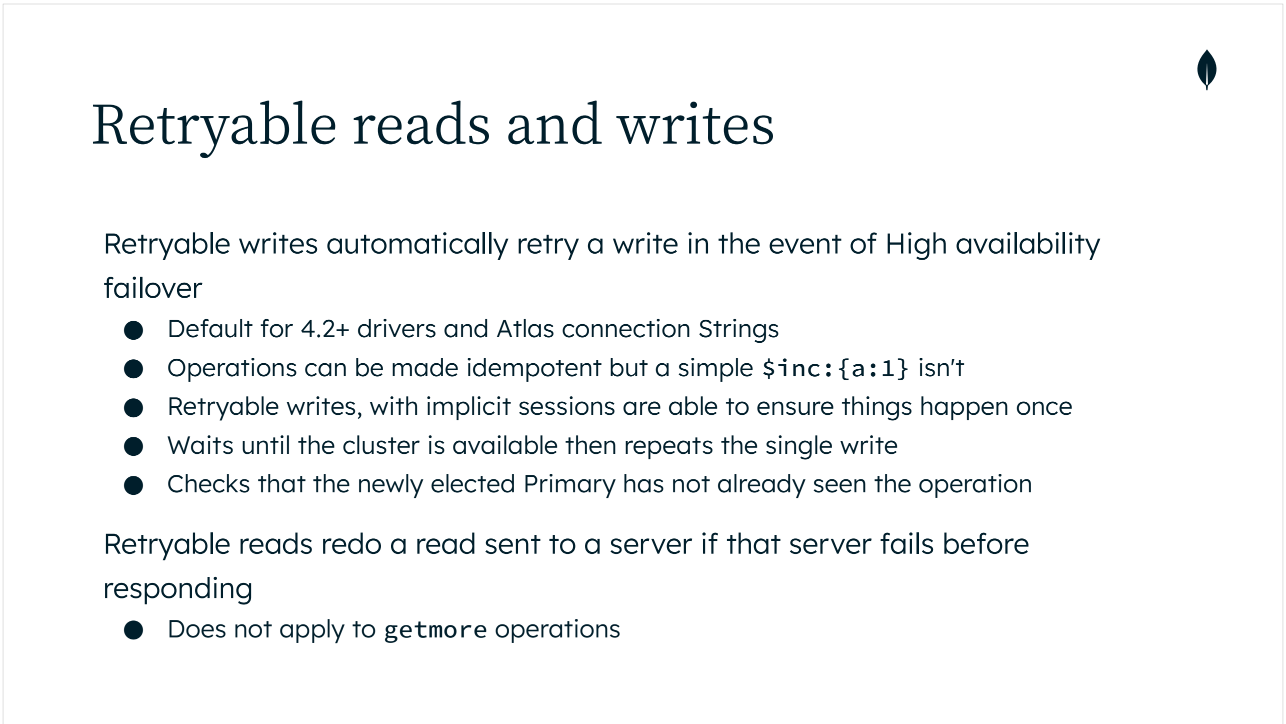
Logical server sessions that can be synchronised between clients

Allow them to have the same causal view of the world

- Allows **Read your own writes**
- One session knows another’s writes have completed
- Allows retryable writes - ability to safely auto-retry after a failover
- Underpin transaction handling

Ultimately sessions revolve around a defined timestamp in the internal operation log

- Sessions are for most developers just an internal concept that require no explicit action.
- Logical server session that keeps track of a given cluster state. Sessions can be coordinated by advancing to a known cluster time. This allows to synchronise multiple processes."
- By passing the timestamp between processes you can ensure that they see the same view of the data, and one will be able to see the operations done by the other.
- They also enable the internal process by which you can safely retry a write after failover and know whether your first attempt had propagated to a replica or not.



Retryable reads and writes

Retryable writes automatically retry a write in the event of High availability failover

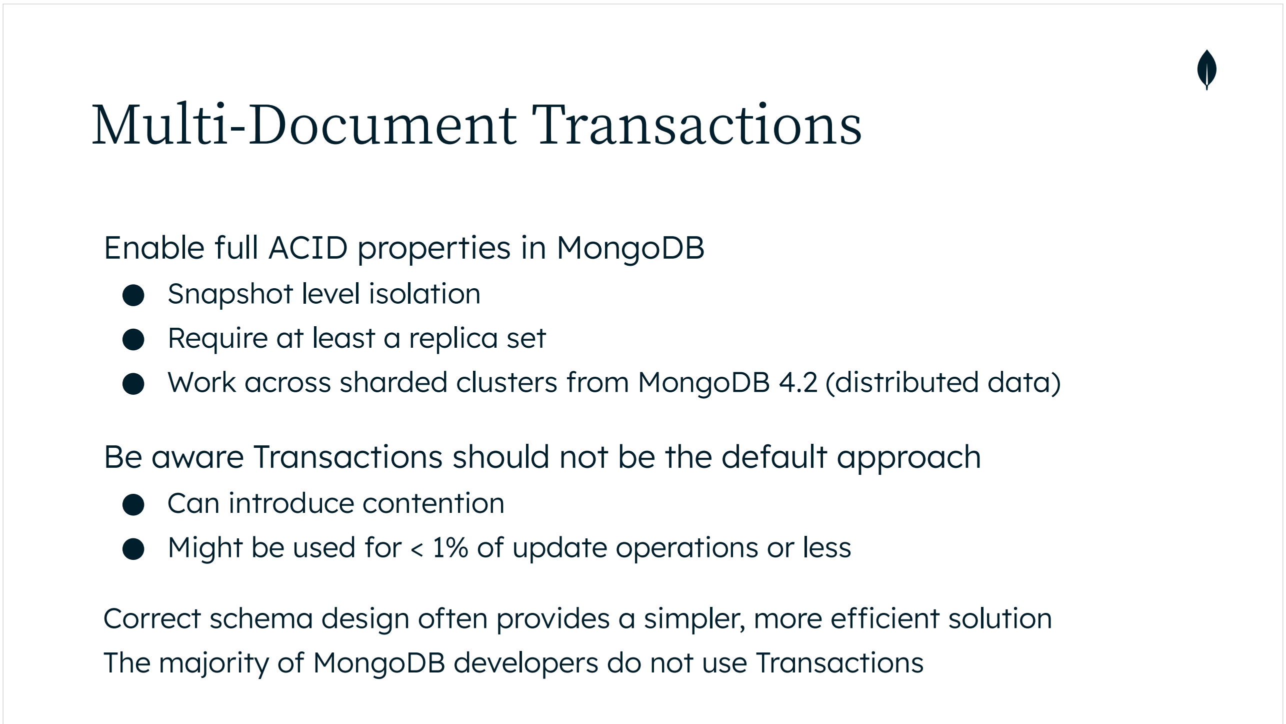
- Default for 4.2+ drivers and Atlas connection Strings
- Operations can be made idempotent but a simple `$inc:{a:1}` isn't
- Retryable writes, with implicit sessions are able to ensure things happen once
- Waits until the cluster is available then repeats the single write
- Checks that the newly elected Primary has not already seen the operation

Retryable reads redo a read sent to a server if that server fails before responding

- Does not apply to `getmore` operations

- Sometimes - we will issue a read or write to the server, and before it completes, that server dies or is killed, or it can become isolated by a network.
- In this case, MongoDB automatically fails over to a new server.
- We could try and handle this in our own code, retrying the operation if it fails with certain events.
- However, the driver can do this for you automatically. As soon as a replacement server is up, it will issue the command again and return. It's smart enough not to do something twice if it had successfully made it to a secondary, and it's already happened after failover. e.g., you don't want to increment a value twice.

Note: In order to get the connection string with retryWrites option true, in Atlas go to your cluster click **Connect > Connect With your Application**.



Multi-Document Transactions

Enable full ACID properties in MongoDB

- Snapshot level isolation
- Require at least a replica set
- Work across sharded clusters from MongoDB 4.2 (distributed data)

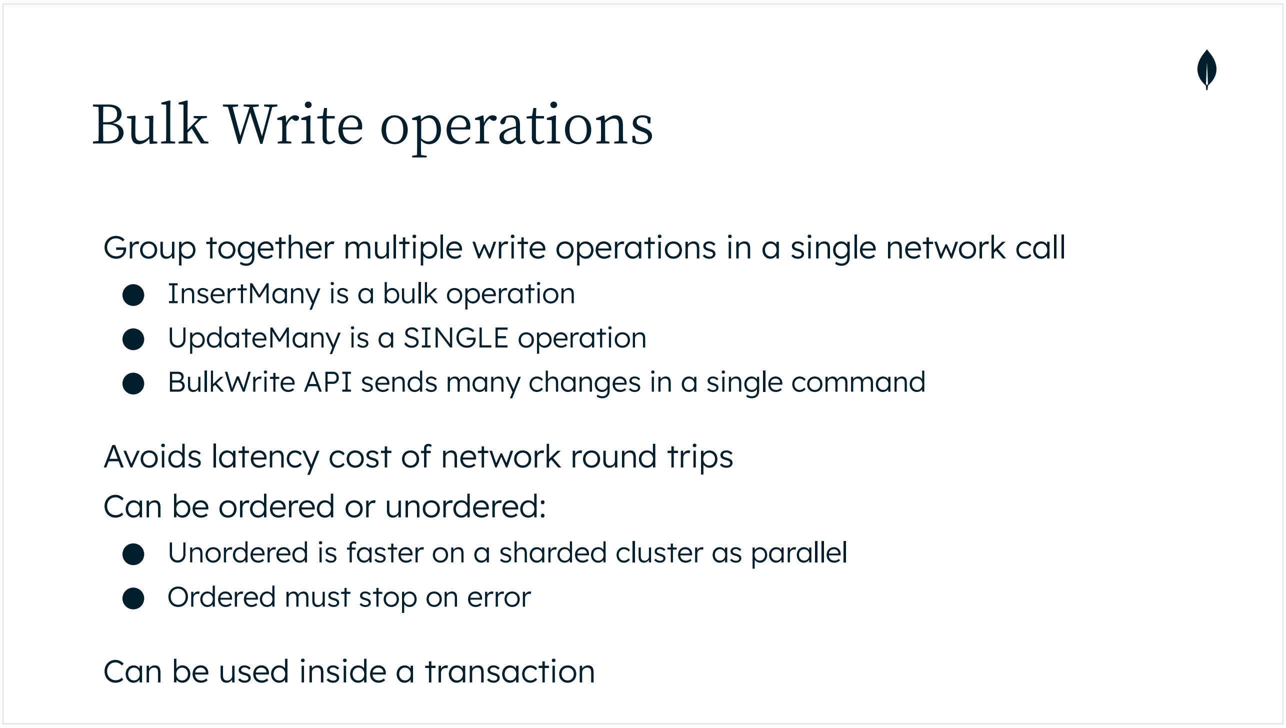
Be aware Transactions should not be the default approach

- Can introduce contention
- Might be used for < 1% of update operations or less

Correct schema design often provides a simpler, more efficient solution

The majority of MongoDB developers do not use Transactions

- ACID stands for Atomicity, Consistency, Isolation, and Durability
- Transactions are a thing that many developers expect.
- MongoDB has good transaction support giving snapshot isolation levels.
- However, a transaction effectively locks a number of records together until committed.
- This sort of locking is one barrier to scale in an RDBMS and one of the reasons for using MDB in the first place.
- Most transactional semantics can be translated to a better lockless pattern, and therefore transactions should be used seldom, if ever.
- The do not change the fundamental that changing multiple documents impacts performance and scaling.



Bulk Write operations

Group together multiple write operations in a single network call

- InsertMany is a bulk operation
- UpdateMany is a SINGLE operation
- BulkWrite API sends many changes in a single command

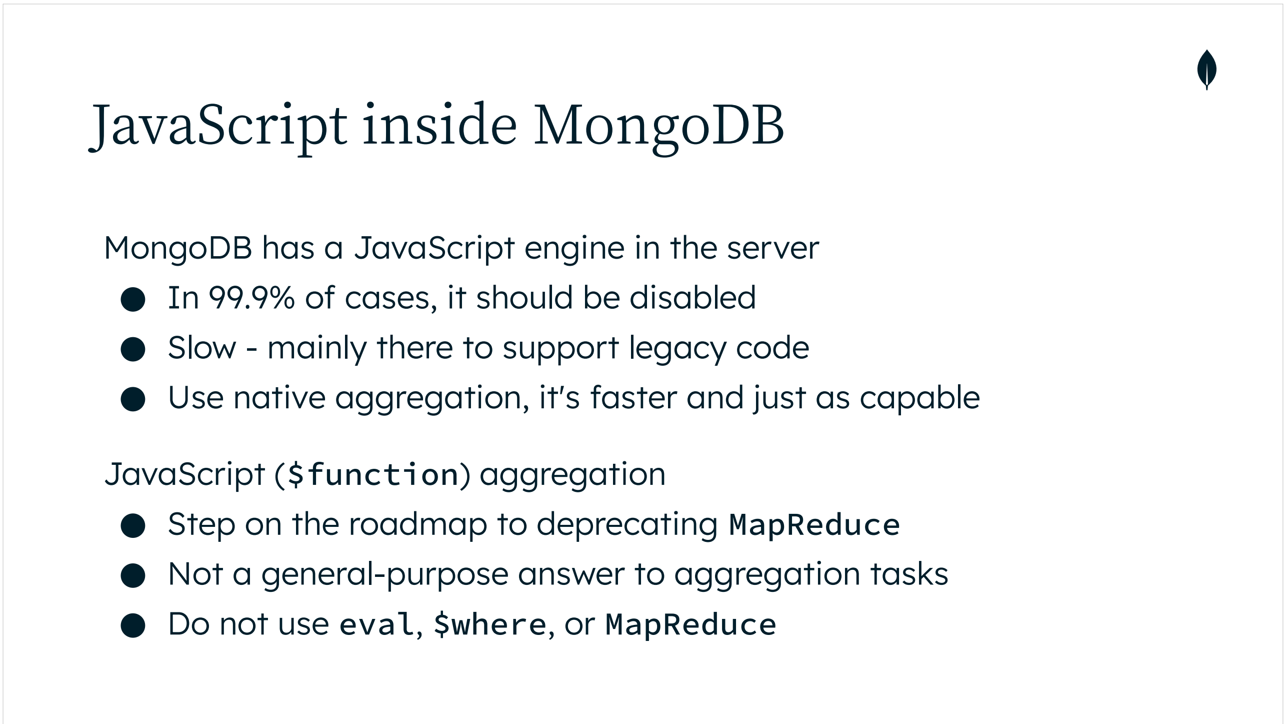
Avoids latency cost of network round trips

Can be ordered or unordered:

- Unordered is faster on a sharded cluster as parallel
- Ordered must stop on error

Can be used inside a transaction

Bulk writes can be grouped together in a single network call.



JavaScript inside MongoDB

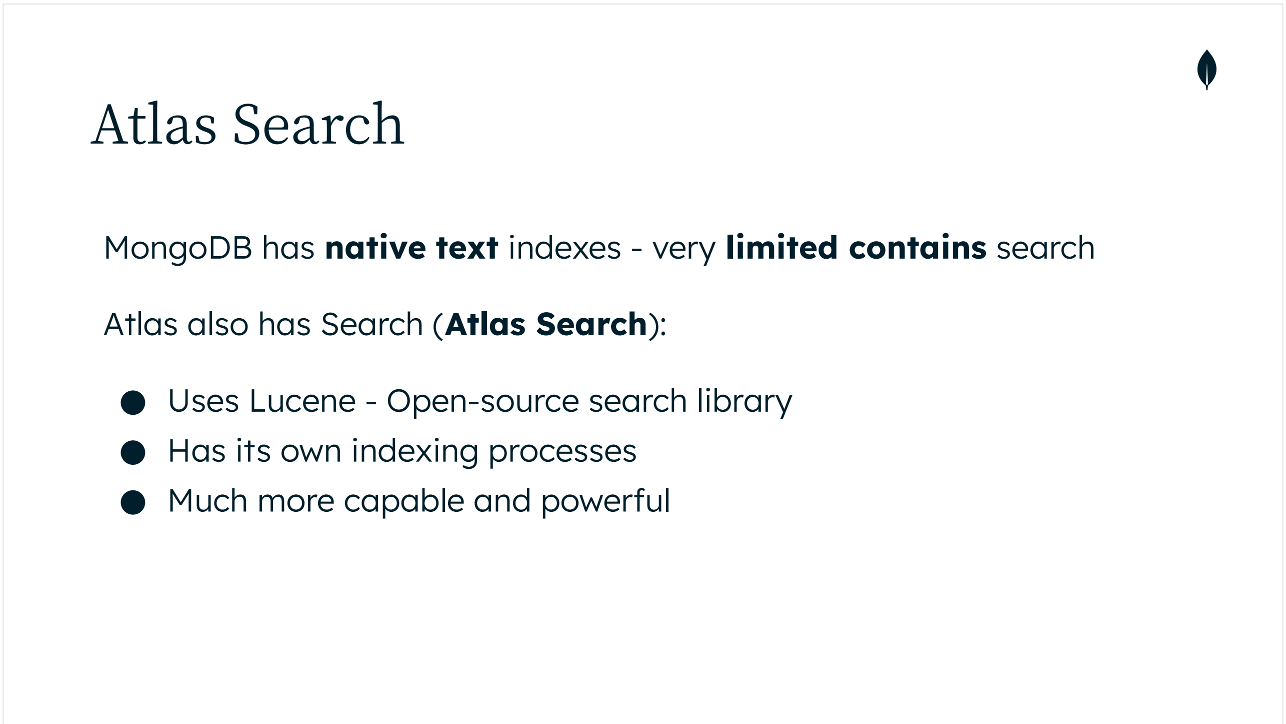
MongoDB has a JavaScript engine in the server

- In 99.9% of cases, it should be disabled
- Slow - mainly there to support legacy code
- Use native aggregation, it's faster and just as capable

JavaScript (`$function`) aggregation

- Step on the roadmap to deprecating `MapReduce`
- Not a general-purpose answer to aggregation tasks
- Do not use `eval`, `$where`, or `MapReduce`

Starting in version 4.2, MongoDB removes the eval command. The deprecated db.eval(), which wraps the eval command, can only be run against MongoDB 4.0 or earlier versions.



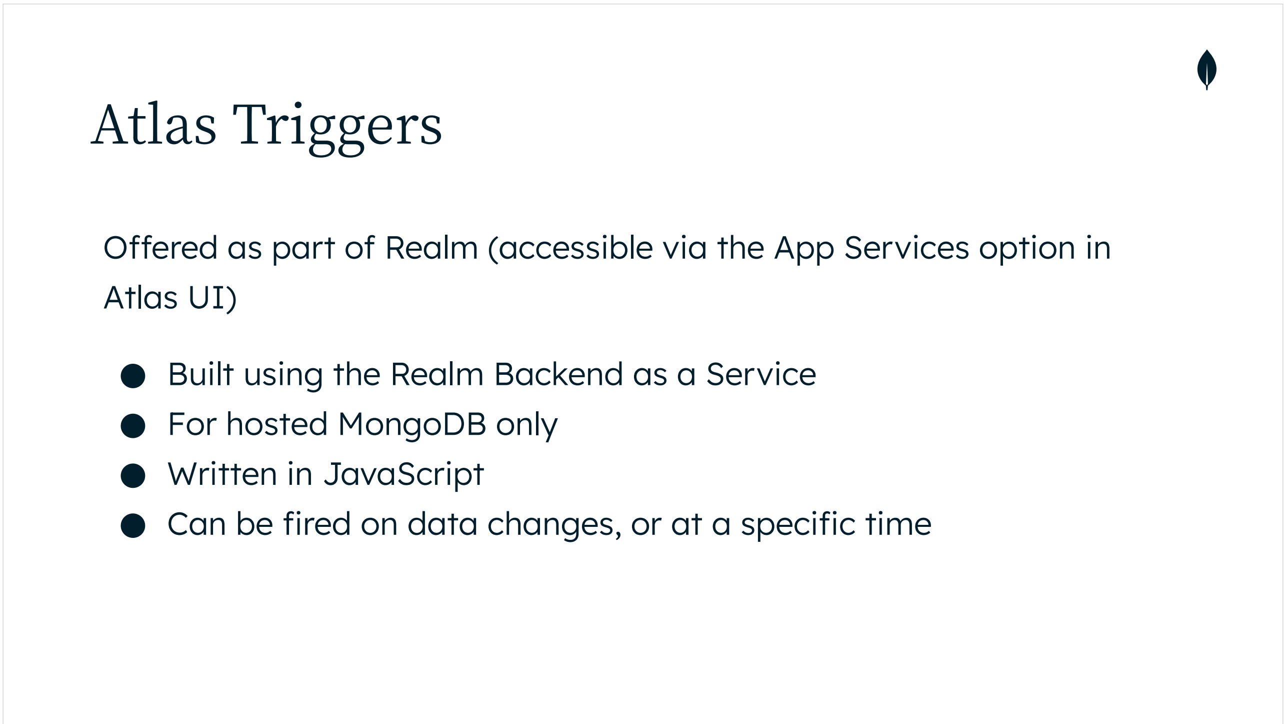
Atlas Search

MongoDB has **native text** indexes - very **limited contains** search

Atlas also has Search (**Atlas Search**):

- Uses Lucene - Open-source search library
- Has its own indexing processes
- Much more capable and powerful

- MongoDB text indexes are limited
- Atlas search should be used where possible as this is based on Lucene and is much more powerful

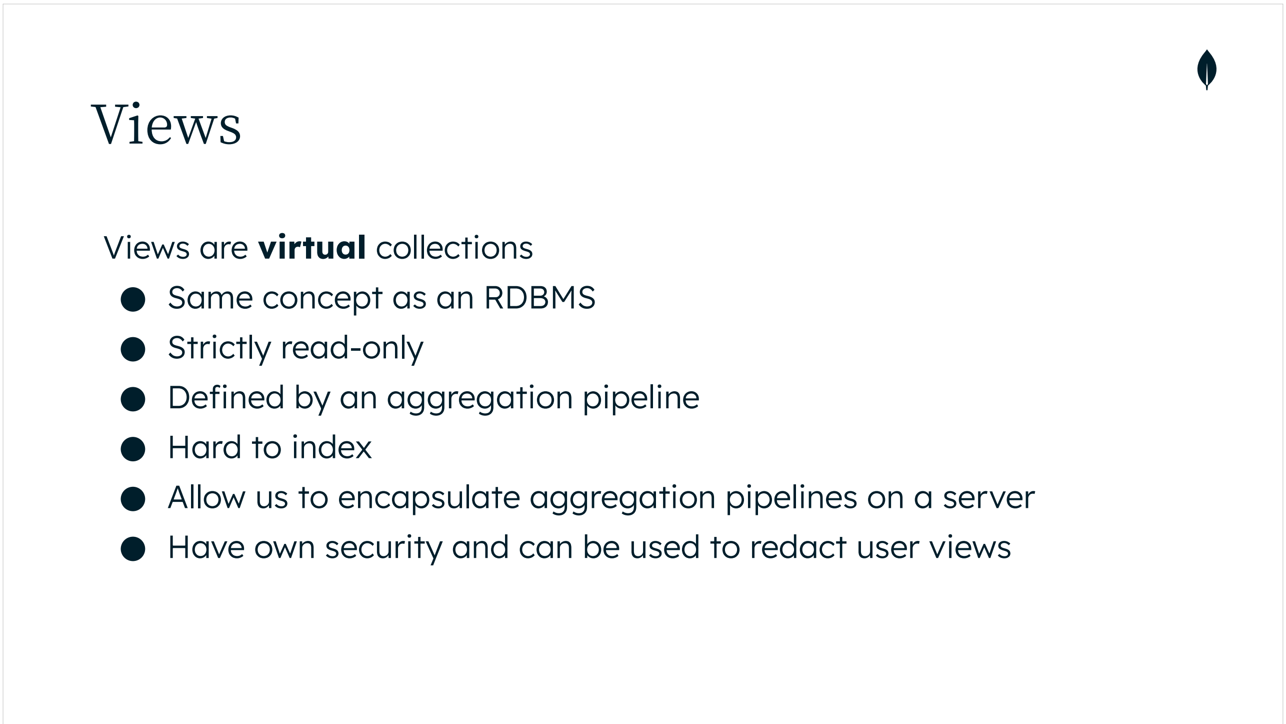


Atlas Triggers

Offered as part of Realm (accessible via the App Services option in Atlas UI)

- Built using the Realm Backend as a Service
- For hosted MongoDB only
- Written in JavaScript
- Can be fired on data changes, or at a specific time

Atlas Triggers are not an integral part of the MongoDB Database Server.



Views

Views are **virtual** collections

- Same concept as an RDBMS
- Strictly read-only
- Defined by an aggregation pipeline
- Hard to index
- Allow us to encapsulate aggregation pipelines on a server
- Have own security and can be used to redact user views

- Views are aggregation pipelines presented as virtual read-only collections.
- When you use them, it adds your operation to the end of the pipeline and runs it.
- They are hard to index as you need to take the pipeline into account - simple projects of a few fields work well to redact collections, though.
- You can do fancy things - however, they aren't as useful as you might first assume.

Quiz Time!

#1. Which of the following statements are true about RegEx in MongoDB?

A RegEx queries can use an index in some cases

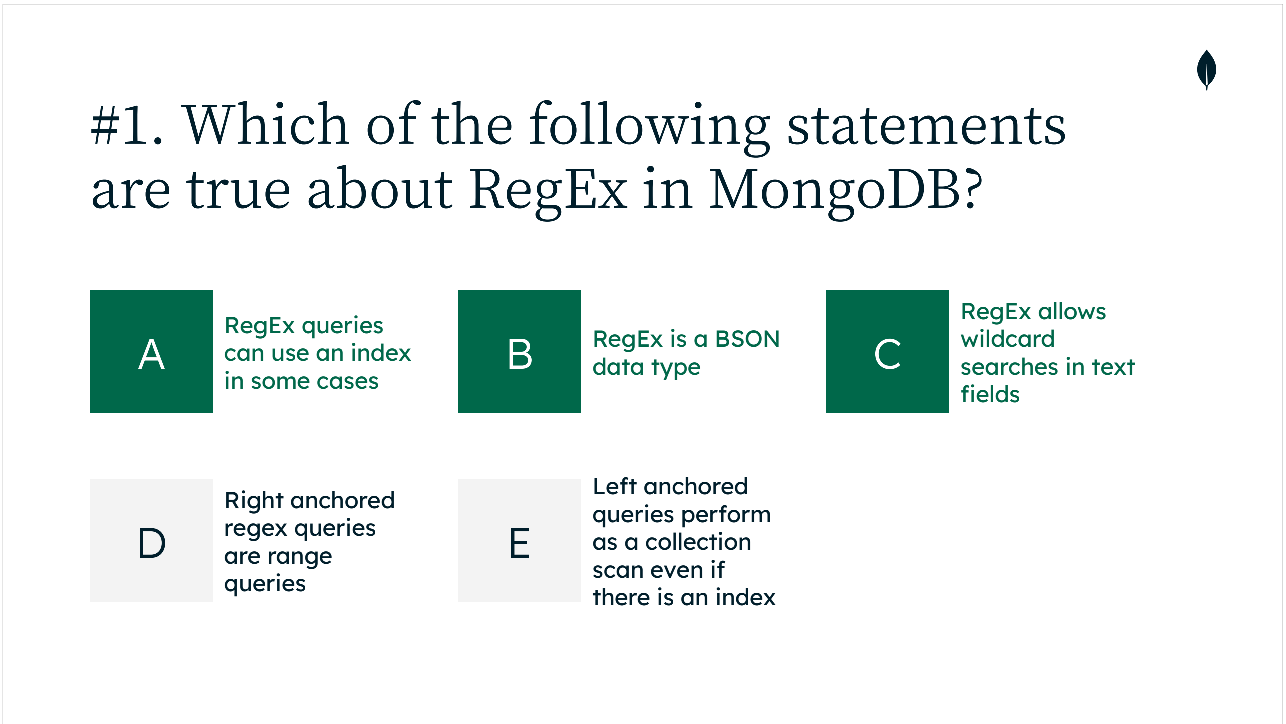
B RegEx is a BSON data type

C RegEx allows wildcard searches in text fields

D Right anchored regex queries are range queries

E Left anchored queries perform as a collection scan even if there is an index

Answer in the next slide.



#1. Which of the following statements are true about RegEx in MongoDB?

- A RegEx queries can use an index in some cases
- B RegEx is a BSON data type
- C RegEx allows wildcard searches in text fields
- D Right anchored regex queries are range queries
- E Left anchored queries perform as a collection scan even if there is an index

Answer: A, B, C

#2. Change Streams can…

A Show when a failover event happens

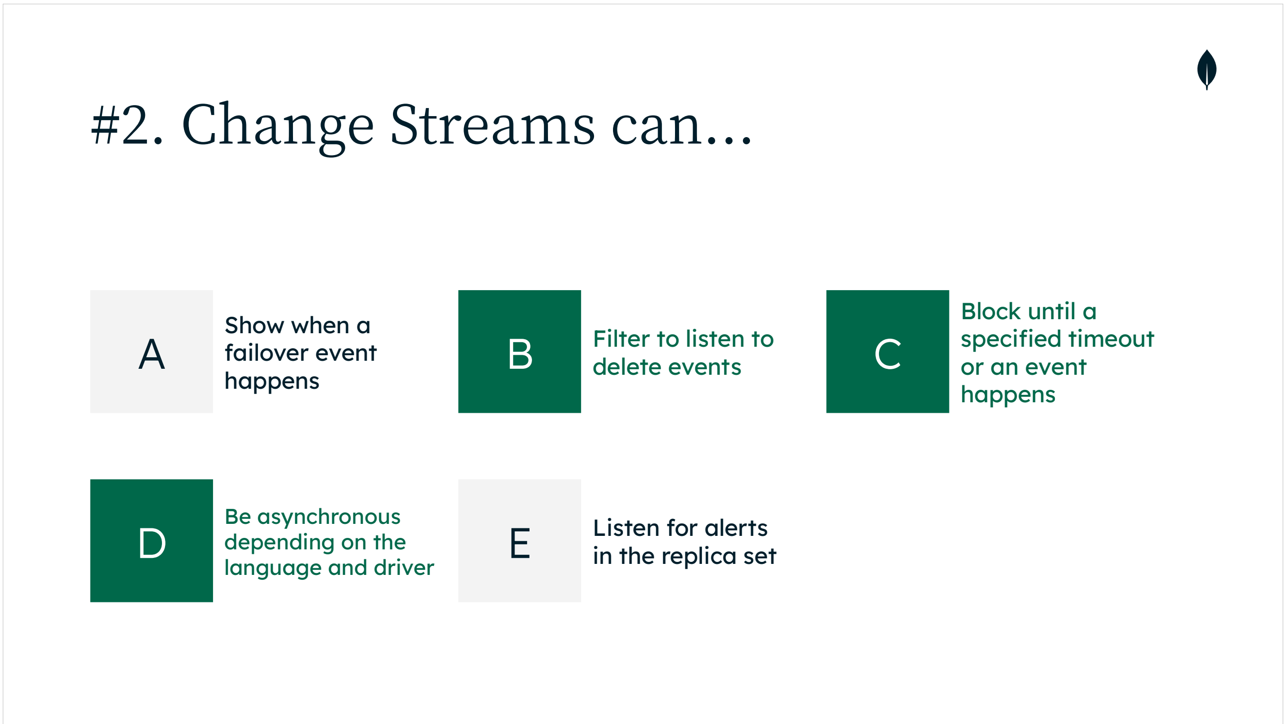
B Filter to listen to delete events

C Block until a specified timeout or an event happens

D Be asynchronous depending on the language and driver

E Listen for alerts in the replica set

Answer in the next slide.



#2. Change Streams can…

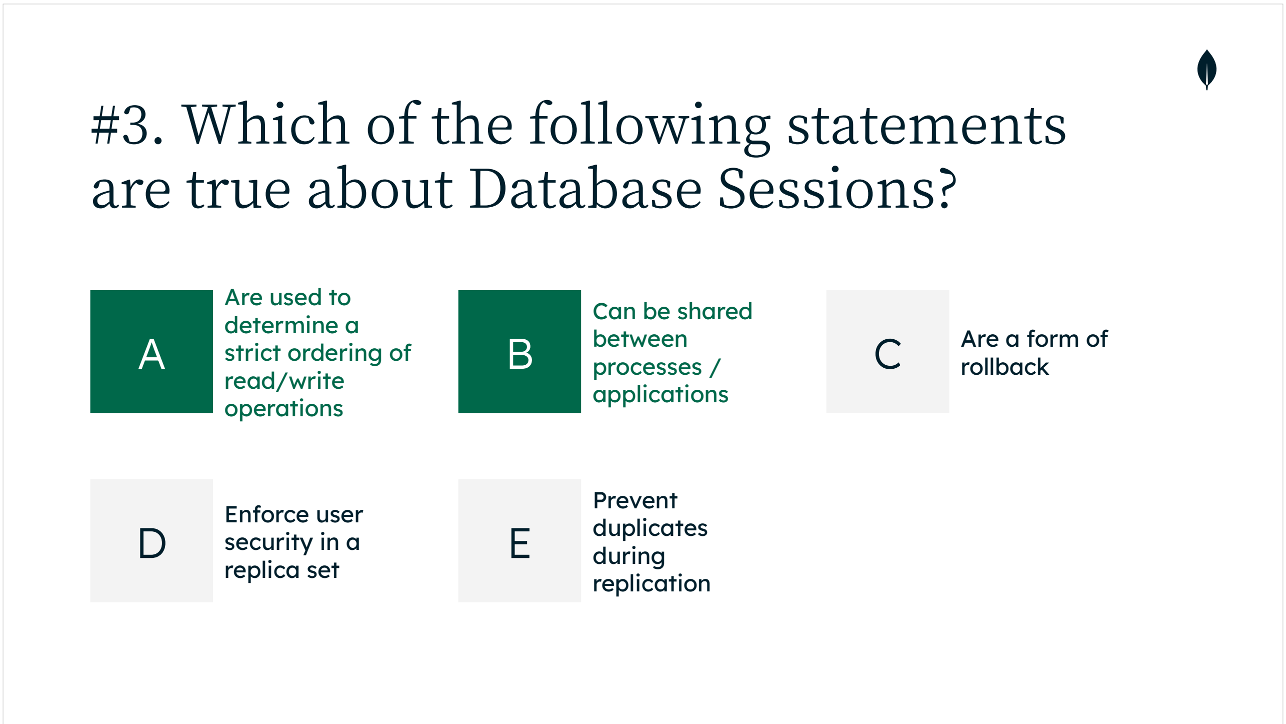
- A Show when a failover event happens
- B Filter to listen to delete events
- C Block until a specified timeout or an event happens
- D Be asynchronous depending on the language and driver
- E Listen for alerts in the replica set

Answer: B, C, D

#3. Which of the following statements are true about Database Sessions?

- A Are used to determine a strict ordering of read/write operations
- B Can be shared between processes / applications
- C Are a form of rollback
- D Enforce user security in a replica set
- E Prevent duplicates during replication

Answer in the next slide.



#3. Which of the following statements are true about Database Sessions?

A Are used to determine a strict ordering of read/write operations

B Can be shared between processes / applications

C Are a form of rollback

D Enforce user security in a replica set

E Prevent duplicates during replication

Answer: A & B

#4. Which of the following statements are true when using retryable writes?

A All operations must be idempotent

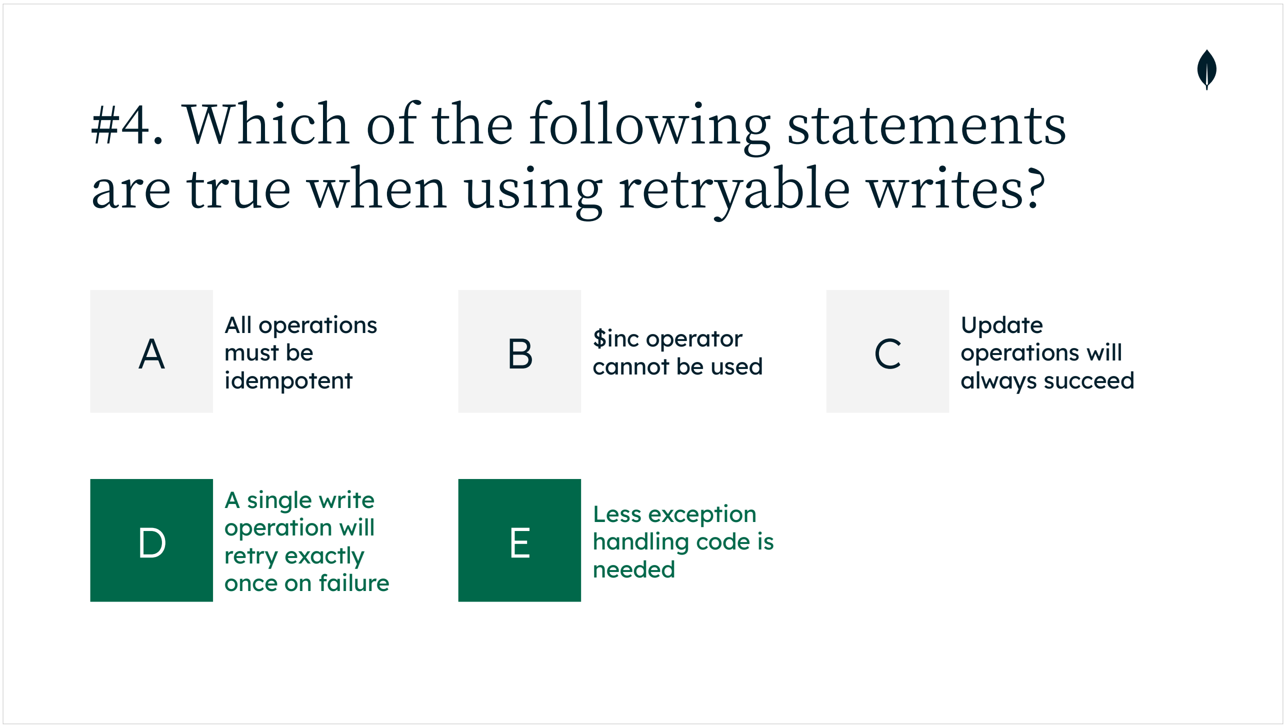
B $inc operator cannot be used

C Update operations will always succeed

D A single write operation will retry exactly once on failure

E Less exception handling code is needed

Answer in the next slide.



#4. Which of the following statements are true when using retryable writes?

- A All operations must be idempotent
- B $inc operator cannot be used
- C Update operations will always succeed
- D A single write operation will retry exactly once on failure
- E Less exception handling code is needed

Answer: D & E

#5. MongoDB Multi Document Transactions …

A Only work on sharded clusters

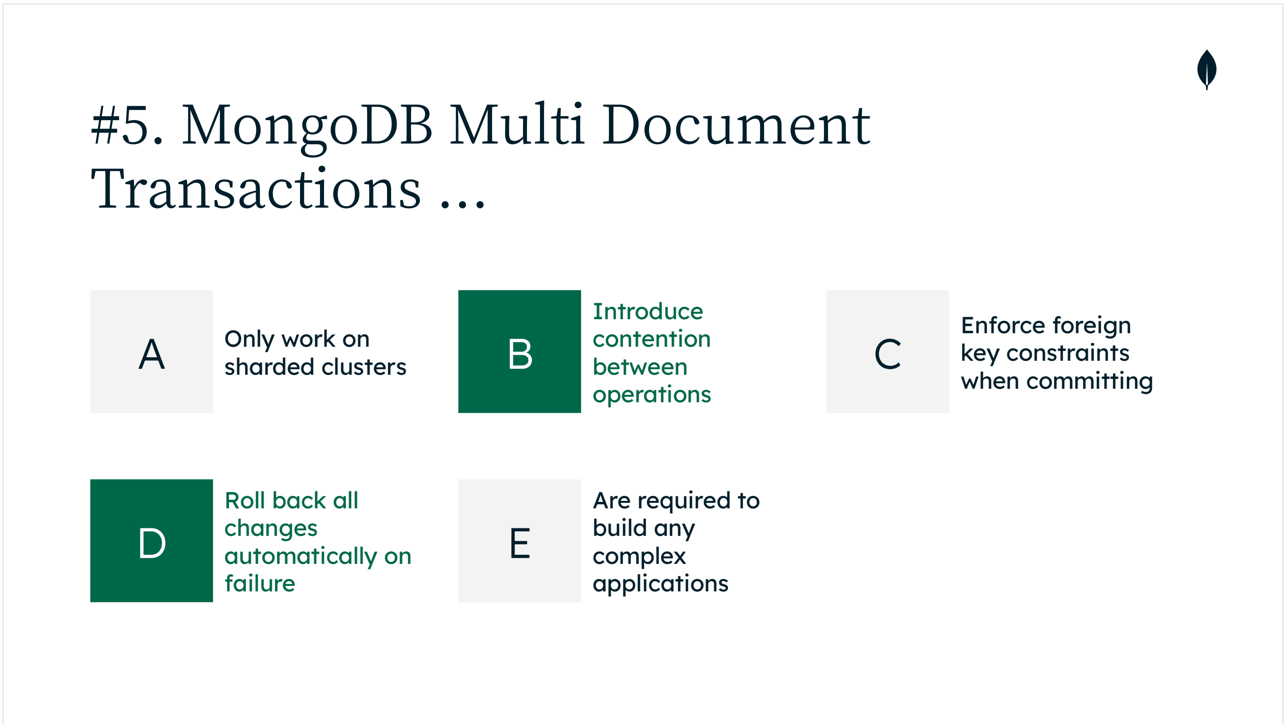
B Introduce contention between operations

C Enforce foreign key constraints when committing

D Roll back all changes automatically on failure

E Are required to build any complex applications

Answer in the next slide.



#5. MongoDB Multi Document Transactions …

- A Only work on sharded clusters
- B Introduce contention between operations
- C Enforce foreign key constraints when committing
- D Roll back all changes automatically on failure
- E Are required to build any complex applications

Answer: B & D

Recap

There is a lot more to MongoDB than simple CRUD:

RegEx, Schema Validation, GridFS, Change Streams, Sessions, Retryable reads & writes, Multi-doc transactions, Bulk write operations, Views, Atlas Search & Atlas Triggers

Appendix

$jsonSchema Example

```
db.createCollection("students", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "address" ],
         properties: {
            name: {
               bsonType: "string",
               description: "must be a string and is required"
            },
            year: {
               bsonType: "int",
               minimum: 2017,
               maximum: 3017,
               description: "must be an integer in [ 2017, 3017 ] and is required"
            },
            major: {
               enum: [ "Math", "English", "Computer Science", "History", null ],
               description: "can only be one of the enum values and is required"
            },
            gpa: {
               bsonType: [ "double" ],
               description: "must be a double if the field exists"
            },
            address: {
               bsonType: "object",
               required: [ "city" ],
               properties: {
                  street: {
                     bsonType: "string",
                     description: "must be a string if the field exists"
                  },
                  city: {
                     bsonType: "string",
                     description: "must be a string and is required"
                  }
               }
            }
         }
      }
   }
})
```