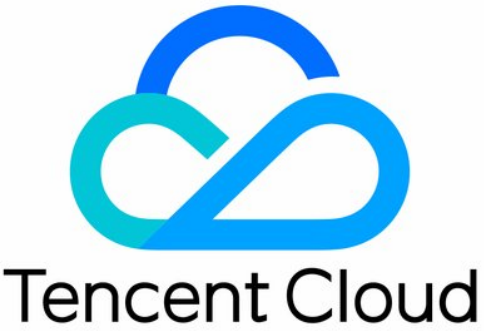
TencentDB for MongoDB Ops and Development Guide



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Ops and Development Guide Development Specifications

Last updated: 2025-02-08 11:16:08

Database Design Specifications

Prohibitions

Prohibit creating too many tables in the database
In the wiretiger engine, each collection needs to create multiple files to store metadata, data, and indexes. Too many small files on the disk can lead to performance degradation. It is recommended to limit the number of tables in a single database to within 100 and the total number of tables in the entire database instance to within 2000.

Recommendations

- Recommend database name starting with db, no special characters except _, all letters in lowercase, and database name not exceeding 64 characters.
- It is recommended that collection names start with t_ and do not contain special characters other than _. The collection name should not exceed 120 characters.

Index design specifications

Prohibitions

- **Do not create indexes in the online database without the background parameter**
  In MongoDB 4.2 and earlier versions, the createIndex() command defaults to foreground mode. In this mode, creating an index will block all database operations, causing business interruptions. When executing createIndex() in online business, be sure to add the background parameter.

  > ⚠ **Note:**
  > The background parameter needs to be in the options parameter of the createIndex() command. Example: `db.test.createIndex({a: 1}, {unique: true, background: true})`. Do not separate different parameters. Example: `db.test.createIndex({a: 1}, {unique: true}, {background: true})`.

- **Sort fields need to be indexed to avoid database OOM (Out Of Memory) caused by extensive in-memory sorting in business operations**

  - For MongoDB 4.2 and earlier versions, a sql statement is allowed to use only 32MB of memory for sorting by default. If exceeded, it will prompt the error `Sort operation used more than the maximum 33554432 bytes of RAM`. At this point, you can adjust the sorting memory by executing `db.runCommand({ getParameter : 1, "internalQueryExecMaxBlockingSortBytes" : 1 } )`.
    However, it is prohibited to adjust this for online business as it increases the likelihood of database OOM. It is recommended to add sort fields to the index to achieve sorting capability through index sorting.
  - For MongoDB 4.4 version, although it provides the option of disk sorting to avoid excessive memory consumption, it is recommended to use index sorting.
- **Prohibit creating too many indexes in a table**
  When MongoDB inserts each piece of data, it also needs to write the index. The more indexes, the higher the cost of writing data. Therefore, the abuse of indexes is prohibited, such as creating an index for each field even if the field will not be queried. It is recommended that a single table should have no more than 10 indexes.

Recommendations

- **It is recommended to regularly clean up unused indexes**
  Indexes bring additional resource consumption during write operations, so it is necessary to streamline indexes as much as possible.
  For MongoDB versions after 4.4, it is recommended to use hidden indexes to hide unused indexes first, and then delete the indexes after confirming that the business is normal.
- **According to the leftmost matching principle, if a single-column index is already included in a composite index, it is recommended to delete it**
  Additional indexes will cause performance waste during write operations.
- **It is recommended to avoid using $ne/$nin operations as much as possible**
  Like other databases (such as MySQL), operations like not equal to or not in cannot effectively utilize indexes and should be avoided as much as possible.
- **It is recommended to create indexes on fields with high distinction**
  If the index field has low distinction, the number of rows scanned in the query will still be large, resulting in low query efficiency and a significant impact on the database load. Therefore, the indexed fields should have high distinction as much as possible.

Database Operation Specifications

Prohibitions

- **Prohibiting the Deployment of SQL Without Confirming the Execution Plan via explain()**
  Before deploying SQL, it is necessary to confirm the execution plan via explain() to ensure it meets expectations; otherwise, deployment may cause failures.
- **In the production environment, disabling authentication is prohibited, especially for databases with open external network access**
  Disabling authentication will expose the database to everyone, especially if the database server has external network access enabled. It is recommended to enable authentication in the production environment. If you must disable authentication, be sure to set firewall rules or an IP allowlist.
- **Storing business data in the admin or local database is prohibited**
  The admin database will lock the db during read and write operations, affecting performance; the local database only saves data locally and does not replicate to secondary nodes, leading to data loss if a master–slave switch occurs. Therefore, using the admin and local databases is prohibited.
- **In a sharded cluster, executing the db.dropDatabase() command and then creating a database with the same name is prohibited**
  - For MongoDB 4.0 and earlier versions, the official documentation requires that after deleting a db and creating a db with the same name, you must execute the restart or flushRouterConfig command on all mongos nodes before business read and write data operations.
  - For MongoDB 4.2, the official documentation requires executing the delete db command again after the first execution, and then executing the restart or flushRouterConfig command on all mongos nodes. For MongoDB 4.4, the official documentation requires executing the delete db command again after the first execution before recreating a database with the same name.
  - For MongoDB 5.0 and above, executing the delete db command once is sufficient.

  Therefore, it is prohibited to directly execute the db.dropDatabase() command and then create a database with the same name in business code. When performing this operation, operations personnel must follow all necessary steps as required by the official documentation.
- **In high concurrency and high performance scenarios, excessive use of in and or is prohibited**
  in or or conditional statements need to be converted into multiple queries at the database level. Excessive in and or operations in high concurrency and high performance scenarios will severely affect request response latency and database load.
- **In high concurrency and high performance scenarios, complex operations should not be delegated to the database**
  MongoDB provides powerful computing capabilities (such as MapReduce), which are very

friendly to developers and greatly reduce business logic. However, these operations inevitably require resources. If complex operations are offloaded to the database layer, it will inevitably place a great burden on the database in high concurrency scenarios. If the database fails, it will cause the entire system to avalanche.
It is recommended to keep database operations simple in high concurrency and high performance scenarios, delegate complex operations to the server, and appropriately add caching in front of the database.

- **In online business, direct batch data remove is prohibited**
  After the remove command is sent to the database, it will first query the _id of records that meet the delete conditions, then delete them one by one according to _id, and record them in the oplog (each deletion will write an oplog).
  When there is a large amount of data meeting the remove conditions, it puts significant pressure on the database and can easily cause a sudden increase in primary-secondary delay.
  For online business, it is recommended to directly use drop collection or use a script to delete records one by one and control the deletion speed. Alternatively, use ttl indexes as much as possible.
- **Customizing the _id field is prohibited in business**_id is the default primary key in MongoDB, and by default, it is an auto-increment sequence. If _id is customized and the business cannot ensure _id increment, the _id index will inevitably need to adjust the B-tree index after each data insertion, which will bring additional burden to the database.
- **In scenarios of direct connection to mongod nodes in a replica set, it is prohibited to configure only a single IP in the connection string; in sharded clusters, it is prohibited to connect to only a single mongos address**
  In online business, if only the primary node of the replica set is connected, a database HA event will cause write interruptions; if only a single mongos is connected, a failure of this mongos will cause business interruptions.
- **Online business is prohibited from setting Write Concern j:false**
  The default Write Concern is generally j:true, meaning the server will write to the journal log before returning to the client side. Generally, do not set j:false, as it may cause data loss if the process suddenly crashes and restarts.
- **Conditionless updates are prohibited in update statements**
  It is recommended to keep multi as the default value (false) to avoid program bugs (such as query parameters being passed as {} due to some exceptions) causing full table data updates.
- **Prohibited from taking out the entire array to update some elements and then writing it back**
  It is recommended to use arrayFilters to modify only the required elements.

Recommendations

- **Suggest partial read and write instead of full read and write**
  In query statements, try to use the $projection operator to project the required fields; in update commands, if only modifying a specific field, it is recommended to use $set, and do not read out the entire document to modify and then write it back in full.
- **Use the db.collection.renameCollection() command with caution in the production environment**
  renameCollection() in versions 4.0 and earlier will block all operations of the db; in versions 4.2 and later, it will block operations of the current and target tables. Additionally, during the execution of renameCollection(), it can cause cursor invalidation, changestream invalidation, and mongodump with the --oplog command to fail. Direct operation during peak periods is prohibited in the production environment.
- **It is recommended to configure WriteConcern for core business with the parameter {w: "majority"}**
  By default, the WriteConcern configuration of most drivers is {w:1}, which considers the request successful after writing to the primary node. If the machine suddenly fails and the written data has not yet been replicated to the secondary node, this configuration can lead to data loss.
  Therefore, for core business in production, it is recommended to configure {w: "majority"}, which will return to the client after the data is synchronized to the majority of nodes. Of course, reliability and performance cannot be balanced; choosing the {w: "majority"} configuration will also increase request latency accordingly.

Recommendations

- **Unless necessary, do not use multi-document transactions extensively in high-performance scenarios**
  Starting from MongoDB 4.0, MongoDB provides multi-document transactions. However, multi-document transactions are only a supplement to database capabilities, and large-scale use of multi-document transactions in high-concurrency, high-performance scenarios requires thorough stress testing.
  Generally, multi-document transactions need to retain snapshots in memory before committing, which may consume a large amount of cache and lead to performance degradation.
- **It is not recommended to use non-persistent connections**
  The authentication logic of MongoDB is a relatively complex calculation process, and by default, MongoDB creates a thread for each connection. A large number of non-persistent connections will impose a significant burden on the database, especially for replica set

clusters without mongos. It is recommended to use persistent connections. For details, refer to the connection pool parameters in the MongoDB URL.

Sharded Cluster Design Specifications

Prohibitions

- **If using _id field as the shard key, do not use range sharding**
  id is by default an increasing sequence and will keep growing with the increase in data volume. If _id is used as the shard key with range sharding, the cluster will continuously balance as data is inserted.
- **Do not directly connect to mongod nodes to write data in a sharded cluster**
  A sharded cluster should write data through mongos. Data written directly through mongod will have no routing information and will be inaccessible.
- **Do not turn off balancer and autoSplit configuration for a long time in the production environment**
  Turning off balance will cause data imbalance between shards, and turning off autoSplit may generate jumbo chunk.
- **Avoid queries without shard key in sharded tables**
  Queries without shard key in sharded tables require scanning all shards and aggregating results in mongos, which is performance-intensive and not recommended.
- **Ensure to set the balancer window in the production environment to avoid affecting business**
  The balance process will put significant pressure on the database, so it is recommended to perform it during the business off-peak period.

Recommendations

- **It is recommended to use fields with high distinction as shard keys, ideally using a unique primary key as the shard key**
  Assume we have a collection storing population information and use gender as the shard key. This shard key is considered to have low cardinality because theoretically, half of the data in the collection will have the same gender field.
  If the shard key has low distinction, it may lead to a large number of records being concentrated on certain shards, and this imbalance cannot be scaled by adding shards. Therefore, it is recommended to use fields with high distinction as shard keys.
- **If using hash sharding, it is recommended to perform pre-allocation, especially for large tables that frequently require bulk data insertion**
  The shardCollection() command by default creates only 2 chunks per shard. As the data volume increases, MongoDB needs to constantly balance and split chunks, which will bring significant burden to the database.

Therefore, for large collections, it is recommended to pre-shard in advance (specify the numInitialChunks parameter in the shardCollection command, with each shard supporting up to 8192 chunks), especially when performing bulk data import into large collections.

Recommendations

- **If there is no strong requirement for scanning in shard key order, it is not recommended to use range sharding, and hash sharding is recommended**
  Range sharding can easily cause imbalance and data hotspots, and because it cannot be pre-sharded, balance is inevitable as data is written. Therefore, it is not recommended unless there is a special range query requirement by shard key.

  > ⚠ **Note:**
  > When using shardCollection, the command sh.shardCollection("records.people", { zipcode: 1 }) indicates range sharding with 1, while the command sh.shardCollection("records.people", { zipcode: "hashed" }) indicates hash sharding with "hashed". Note not to mix them up.

- **It is not recommended to use non-sharded tables in a sharded cluster**
  If the shardCollection command is not executed in a MongoDB sharded cluster, data is stored on the primary shard by default. A large number of non-sharded tables can cause data volume inconsistency between shards. Over long-running periods, this may result in some shards having significantly more data, potentially filling up the disk. In such cases, operations staff have to manually migrate data using movePrimary, which increases operational complexity.

Command Support in Sharded Cluster v3.2

Last updated：2025-02-08 11:16:36

Sharding Strategy

- Ranged-based sharding is supported.
- The shardkey is indexed compound fields.
- Sharding is required for all data sets in a shard instance. It is recommended to store non-sharded data in a separate replica set instance.

Authentication Mechanism

MongoDB is fully compatible with SCRAM-SHA-1 and MONGODB-CR.

Supported Sharded Cluster Commands

Classify	Command	Subcommand	System Support
Basic CRUD commands	find	filter	Supported
		sort	Supported
		projection	Supported
		hint	Supported
		skip	Supported
		limit	Supported
		batchSize	Supported
		singleBatch	Supported

comment	Supported
maxScan	Supported
maxTimeMS	Not supported.
readConcern	Supported
max	Supported
min	Supported
returnKey	Supported
showRecordId	Supported
snapshot	Not supported.
tailable	Not supported.
oplogReplay	Not supported.
noCursorTimeout	Supported
awaitData	Not supported.
allowPartialResults	Not supported.

insert	The `shardkey` field is required and must be the same for batch INSERT operations	Supported
update	The updated field cannot be `shardkey`	Supported
delete	–	Supported
findandmodify	–	Supported
count	–	Supported
distinct	The `shardkey` field is required	Supported
aggregate	–	Supported
group	–	Not supported.
mapReduce	–	Not supported.
getmore	–	Supported
getLastError	–	Not supported.
getPrevError	–	Not supported.
resetError	–	Not supported.
eval	–	Not supporte

			d.
	geoNear	–	Not supported.
	geoSearch	–	Not supported.
	parallelCollectionScan	–	Not supported.
Diagnostic commands	collStats	–	Supported
	dbstats	–	Supported
	explain	–	Supported
	listDatabases	–	Supported
	serverStatus	–	Not supported.
	top	–	Not supported.
Sharding commands	enableSharding	–	Supported
	shardCollection	–	Supported
Management commands	listCollections	–	Supported
	dropDatabase	–	Supported
	drop	–	Supported

creareIndexes	–	Supported
listIndexes	–	Supported
dropIndexes	–	Supported
logout	–	Supported
renameCollection	–	Not supported.
copydb	–	Not supported.
create	–	Not supported.
clone	–	Not supported.
cloneCollection	–	Not supported.
cloneCollectionAsCapped	–	Not supported.
convetToCapped	–	Not supported.
filemd5	–	Not supported.
fsync	–	Not supported.

clean	–	Not supported.
connPoolSync	–	Not supported.
connectionStatus	–	Not supported.
compact	–	Not supported.
collMod	–	Not supported.
reIndex	–	Not supported.
setParameter	–	Not supported.
getParameter	–	Not supported.
repairDatabase	–	Not supported.
repairCursor	–	Not supported.
touch	–	Not supported.
shutdown	–	Not supported.

	logrotate	–	Not supported.
	killop	–	Not supported.
User management commands	–	–	Not supported.
Role management commands	–	–	Not supported.
Replica set commands	–	–	Not supported.

Command Support in v3.6

Last updated: 2025-02-08 11:17:00

For a list of MongoDB's commands, please see the official documentation.
TencentDB for MongoDB v3.6 does not support the following commands:

Command Classification	Unsupported Commands
Sharding Commands	addShard
	removeShard
Query and Write Operation Commands	getPrevError
Role Management Commands	dropAllRolesFromDatabase
Replication Commands	replSetAbortPrimaryCatchUp
	replSetFreeze
	replSetGetConfig
	replSetGetStatus
	replSetInitiate
	replSetMaintenance
	replSetReconfig
	replSetResizeOplog
	replSetStepDown
	replSetSyncFrom
	resync
Administration Commands	cloneCollection
	cloneCollection
	cloneCollectionAsCapped
	compact

	connPoolSync
	logRotate
	setParameter
	shutdown
Diagnostic Commands	availableQueryOptions
	dbHash
	getCmdLineOpts
	getLog
	shardConnPoolStats
System Events Auditing Commands	logApplicationMessage