

## SequoiaDB Information Center

# Contents

SequoiaDB Database Description.....	6
Features.....	6
Data Model.....	7
Infrastructure.....	9
Database Concept.....	14
Database.....	14
Document.....	14
Collection.....	17
Collection Space.....	18
Database Server.....	19
Index.....	19
Aggregate.....	21
Transaction.....	22
Eventually Consistent.....	22
Examples.....	23
Read/Write Splitting.....	23
Background Task.....	23
Cluster.....	24
Mode.....	24
Node.....	25
ReplicaGroup.....	32
Sharding.....	35
Domain.....	41
Install Overview.....	42
Overview on Database Deployment.....	42
Easiest deployment.....	42
High-availability Deployment.....	43
High-performance Deployment.....	44
System Requirement of SequoiaDB Installation.....	45
Hardware Requirement.....	45
Supported Operating System.....	46
Software Requirement.....	46
Install Media.....	48
Installation of SequoiaDB Server.....	48
Installing Linux version on SequoiaDB Server.....	49
Start SequoiaDB web service.....	50
System configuration and startup.....	50
Configuration and startup of standalone mode.....	51
Configuration and startup of cluster mode.....	51
The Test Environment.....	54
Uninstall.....	54
DataBase Administration.....	56
DataBase Management.....	56
Database Runtime Configurion.....	56
Monitoring.....	59
Engine Dispatchable Unit.....	83
Log.....	84
DataBase Tool.....	85
Cluster Management.....	90
Add master to cluster.....	90

Catalog Shardings Management.....	90
Data Sharding Management.....	91
Create coord node.....	92
Sample.....	93
Operation and Maintain.....	95
Start and stop the cluster.....	96
Data backup.....	97
View backup information.....	98
Data Recovery.....	100
Recovery.....	101
Monitor.....	101
System Security.....	103
<b>Hadoop Integration.....</b>	<b>104</b>
SequoiaDB with Hadoop deployment.....	104
With MapReduce Integration.....	104
Integration with Hive.....	107
SequoiaDB list of the supported versions Hive.....	108
Configuration method.....	108
Use.....	108
Integration with Pig.....	109
<b>Development Instruction.....</b>	<b>110</b>
SequoiaDB shell.....	110
SequoiaDB Shell Introduction.....	110
Tips when using shell.....	111
Basic Manipulation in Shell.....	113
Create.....	113
Read.....	115
Update.....	118
Delete.....	119
SequoiaDB Application Development.....	119
C Driver.....	119
C++ Driver.....	127
Java Driver.....	134
PHP Driver.....	142
C# Driver.....	146
BSON Interface to C.....	151
BSON Interface to C++.....	153
C BSON API.....	154
C++ BSON API.....	154
SequoiaDB Cluster Manager.....	154
<b>Reference.....</b>	<b>156</b>
SequoiaDB JavaScript Method List.....	156
db.backupOffline().....	158
db.cancelTask().....	159
db.createCataRG().....	160
db.createCS().....	161
db.createDomain().....	162
db.createProcedure().....	163
db.createRG().....	164
db.createUsr().....	164
db.dropCS().....	165
db.dropDomain().....	166
db.dropUsr().....	167
db.eval().....	167
db.exec().....	168

db.execUpdate()	169
db.flushConfigure()	169
db.getCS()	170
db.getDomain()	170
db.getRG()	171
db.list()	172
db.listBackup()	173
db.listCollections()	174
db.listCollectionSpaces()	175
db.listDomains()	175
db.listProcedures()	176
db.listReplicaGroups()	177
db.listTasks()	178
db.removeBackup()	179
db.removeProcedure()	180
db.removeRG()	180
db.resetSnapshot()	181
db.setSessionAttr()	181
db.snapshot()	182
db.startRG()	185
db.transBegin()	185
db.transCommit()	186
db.transRollback()	186
db.waitTasks()	187
db.collectionspace.createCL()	187
db.collectionspace.dropCL()	189
db.collectionspace.getCL()	190
db.collectionspace.collection.aggregate()	190
db.collectionspace.collection.alter()	193
db.collectionspace.collection.attachCL()	194
db.collectionspace.collection.count()	194
db.collectionspace.collection.createIndex()	195
db.collectionspace.collection.detachCL()	196
db.collectionspace.collection.dropIndex()	196
db.collectionspace.collection.find()	197
db.collectionspace.collection.getIndex()	198
db.collectionspace.collection.insert()	198
db.collectionspace.collection.listIndexes()	200
db.collectionspace.collection.remove()	200
db.collectionspace.collection.split()	201
db.collectionspace.collection.splitAsync()	202
db.collectionspace.collection.update()	203
db.collectionspace.collection.upsert()	204
cursor.close()	206
cursor.current()	207
cursor.hint()	207
cursor.limit()	208
cursor.next()	209
cursor.size()	209
cursor.skip()	210
cursor.sort()	210
cursor.toArray()	211
rg.createNode()	212
rg.getDetail()	212
rg.getMaster()	214
rg.getNode()	214

rg.getSlave().....	215
rg.removeNode().....	215
rg.start().....	216
rg.stop().....	216
node.connect().....	217
node.getHostName().....	217
node.getNodeDetail().....	218
node.getServiceName().....	218
node.start().....	218
node.stop().....	219
domain.alter().....	219
domain.listCollections().....	220
domain.listCollectionSpaces().....	220
Operator.....	221
Match Operator.....	223
Update Operator.....	233
Aggregate Operator.....	238
SQL Grammar.....	246
sql create collectionspace.....	247
sql drop collectionspace.....	248
sql create collection.....	248
sql drop collection.....	248
sql create index.....	249
sql drop index.....	249
sql list collectionspaces.....	250
sql list collections.....	250
sql insert into.....	250
sql select.....	251
sql update.....	252
sql delete.....	252
sql group by.....	253
sql order by.....	253
sql limit.....	254
sql offset.....	254
sql as.....	254
sql inner join.....	255
sql left outer join.....	255
sql right outer join.....	256
sql sum().....	256
sql count().....	256
sql avg().....	257
sql max().....	257
sql min().....	257
sql first().....	258
sql last().....	258
sql push().....	258
sql addtoiset().....	259
sql buildobj().....	259
sql mergearrayset().....	260
Mapping Table from SQL to SequoiaDB.....	260
Limits.....	262
Error Code List.....	263

## SequoiaDB Database Description

---

SequoiaDB database is a kind of new corporation distributed non-relation database. It helps corporation to reduce IT cost, and provides a steady, dependable, efficient and flexible underlying platform.

### Advantage

- Through non-structure storage and distributed processing, SequoiaDB provides near-linear scalability without caring about underlying storage limit.
- SequoiaDB provides high-useability accurate to sharding level, prevents the system from going down caused by problems of server, machine room or human false, and keeps data availibale on line in 24x7 hours.
- SequoiaDB provides complete corporation functionalities and enables users to conveniently manage high-concurrent tasks and big data analysis programs.
- Hasing strengthened non-relation data module, SequoiaDB helps corporations to rapidly develop and deploy applications in order to achieve the high-flexiblity of applications.
- SequoiaDB guarantees data eventual consistency, which eradicates data loss.
- In SequoiaDB, on-line applications and big data analysis programs share the same background database at the same time. In SequoiaDB, data analysis programs and on-line application in the same system are invisible to each other because of read/write splitting.

## Features

---

With corporation requirements being more and more complex and diverse, and business being rapidly expanded and producing vast data, IT departments need to provide their users with more and more information. Meanwhile under the current background of economy, IT department needs to not only offer efficient service but also reduce the cost of device and program maintenance.

SequoiaDB database provides massive cluster data platform based on PC server, enables IT departments to provide steady, dependable and efficient data service, and greatly reduce the cost of development, device and program maintenance of applications in IT departments.

Through deploying and using SequoiaDB database, users can gain:

### Horizontal Scalability

Traditional relation database cannot acheive horizontal scalability, but SequoiaDB can perfectly solve this problem. Through vertical splitting of data and the application of non-relation data model, SequoiaDB greatly break through the bottleneck of massive data exchange within shardings in relation database. In this way, it acheive horizontal scalability.

### Permanent High-useability

SequoiaDB stores multiple real-time copies of every piece of data, in order to prevent the system from going down caused by problems of server, machine room or human false, and keeps data availibale on line in 24x7 hours.

### Complete Corporation Support

SequoiaDB provides corporation with user-friendly and complete management, maintenance and monitoring user interface, and 24x7-hour telephone and present technical support.

### Strengthened Non-relation Data Module

SequoiaDB uses the structure called JSON, flexibly simplifies the data maintenance in relation database. In SequoiaDB, data is stored in type of object according to the requirement of applications, which greatly reduce the cost of developmetn and maintenance of application.

### Data Eventual Consistency

SequoiaDB guarantees data eventual consistency in the massive distributed infrastructure, which meets users' requirement of real-time manipulation and data consistency.

### Combination of On-line Applications and Big Data Analysis

In SequoiaDB, data analysis programs and on-line application in the same system are invisible to each other because of read/write splitting. Big data analysis is acheive through Hadoop technology.

SequoiaDB is the first corporation file-based non-relation database in the world. It is aimed at serving users with a distributed data platform with high-scability, high-useability, high-quality and easy to maintain. SequoiaDB meets users' needs of big data analysis and low cost.

## Data Model

---

SequoiaDB use the data model called JSON rather than traditional relation data module.

The full name of JSON is Javascript Object Notation. It is a kind of lightweight data-exchange format. It is easy for human to read and write JSON. It is also easy for machine to generate and parse JSON.

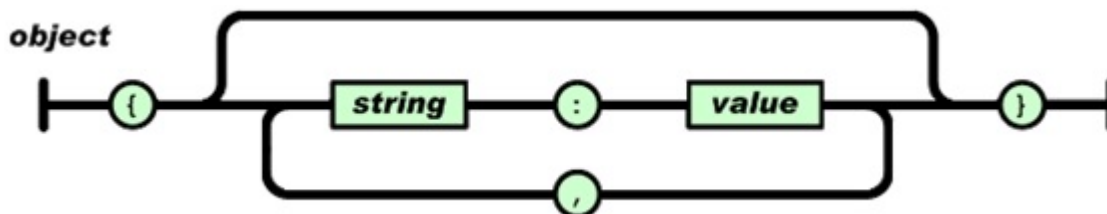
It is based on a subset of JavaScript Programming Language, Standard ECMA-262 3rd Edition – December 1999. It is in type of text. JSON supports nesting structure and array.

The generation of JSON is based on two kinds of structure:

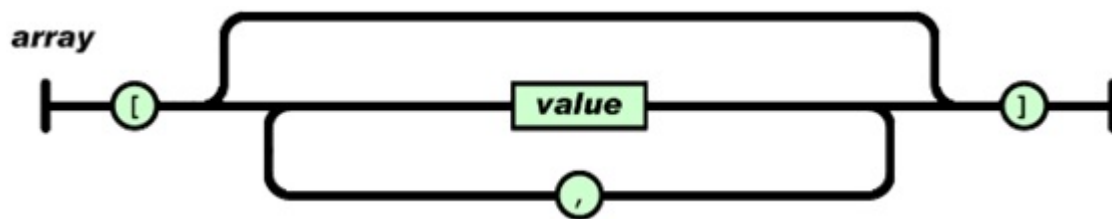
- Key-value pair collection. In the structure of key-value pair collection, aevery data element contains a name and a value. The value in it can be in the type of figure, string and other common types, or nested JSON object and array.
- Array. Every element in a array doesn't contain element name. The value in it can be in the type of figure, string and other common types, or nested JSON object and array.

The format of JSON may be:

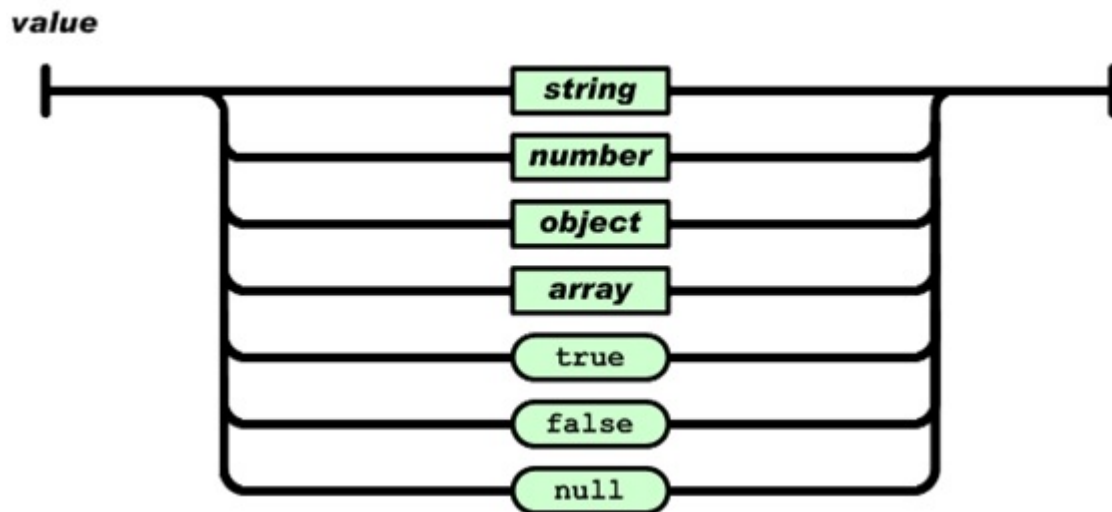
- The object is a unordered "key-value pair" collection. It begins with "{" (left brace), and ends with "}" (right brace). Every element name is followed with a ":" (colon). Elements are seperated with "," (comma).



- Array is a ordered value collection. It begins with a "[" (left bracket), and ends with "]" (right bracket). Values are seperate with "." (comma).



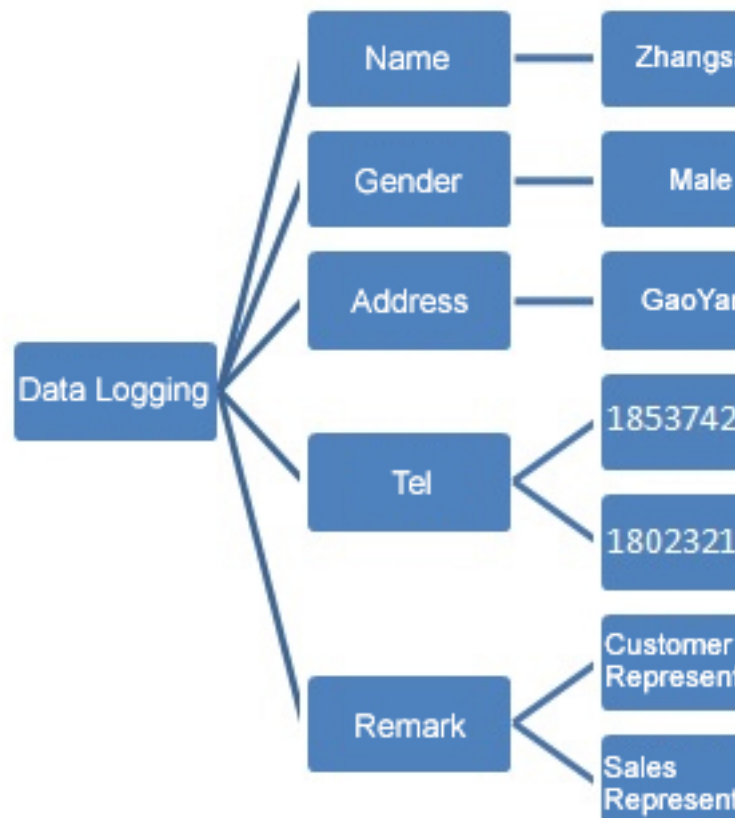
- Value is in type of string, figure, true, false, null, object, array, and special data structures in SequoiaDB (such as date, time .etc). It is surrounded with double quotations.



A typical nesting structure is as follow:

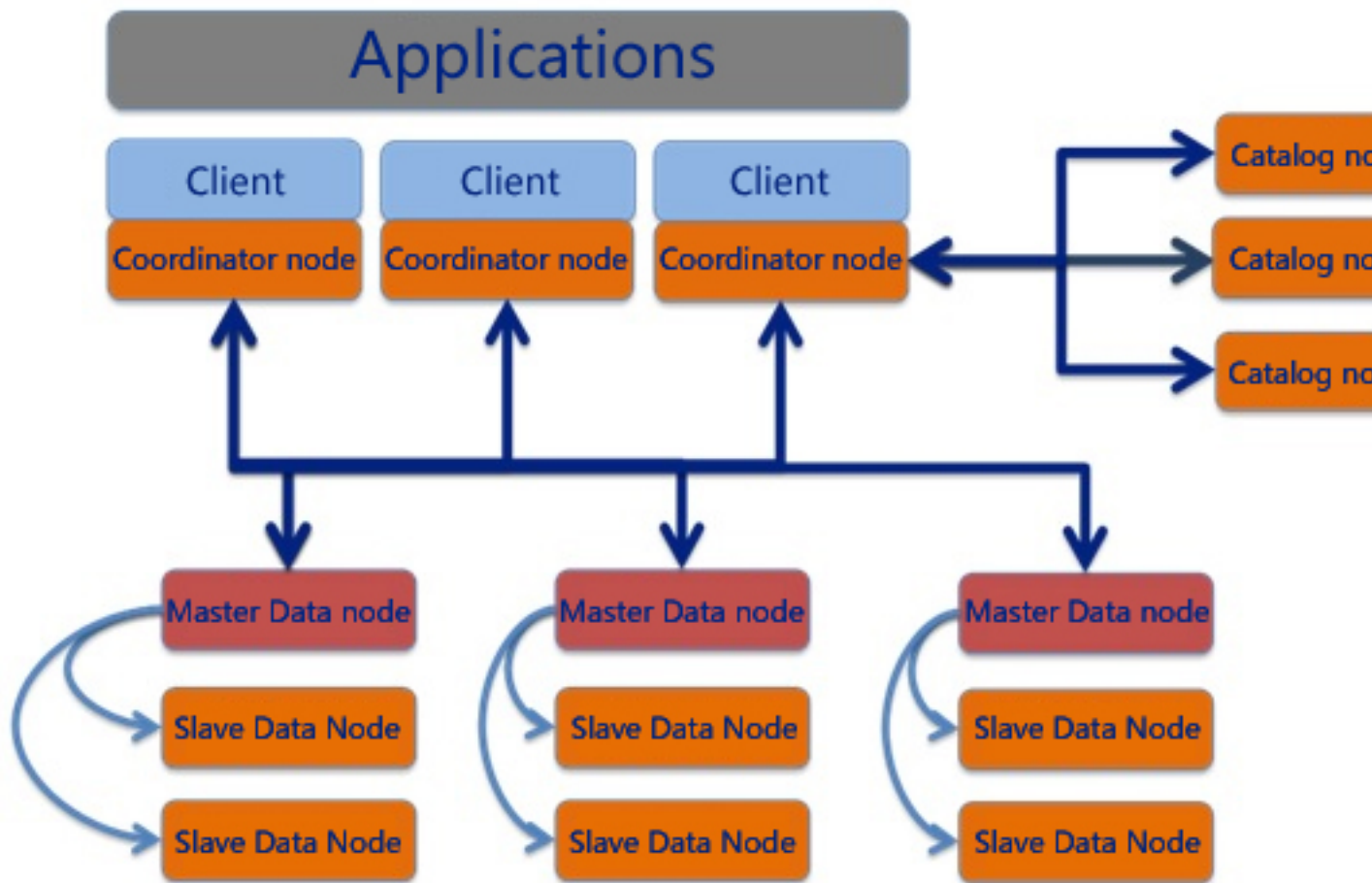


```
{
  "name": "Zhangsan",
  "gender": "male",
  "address": "GaoYang"
  "tel": [
    1853742xxx,
    1802321xxx
  ],
  "remark": [
    "Customer Representative",
    "Sales Representative"
  ]
}
```



## Infrastructure

SequoiaDB applies distributed structure. The picture below describes a general summary of SequoiaDB system architecture.



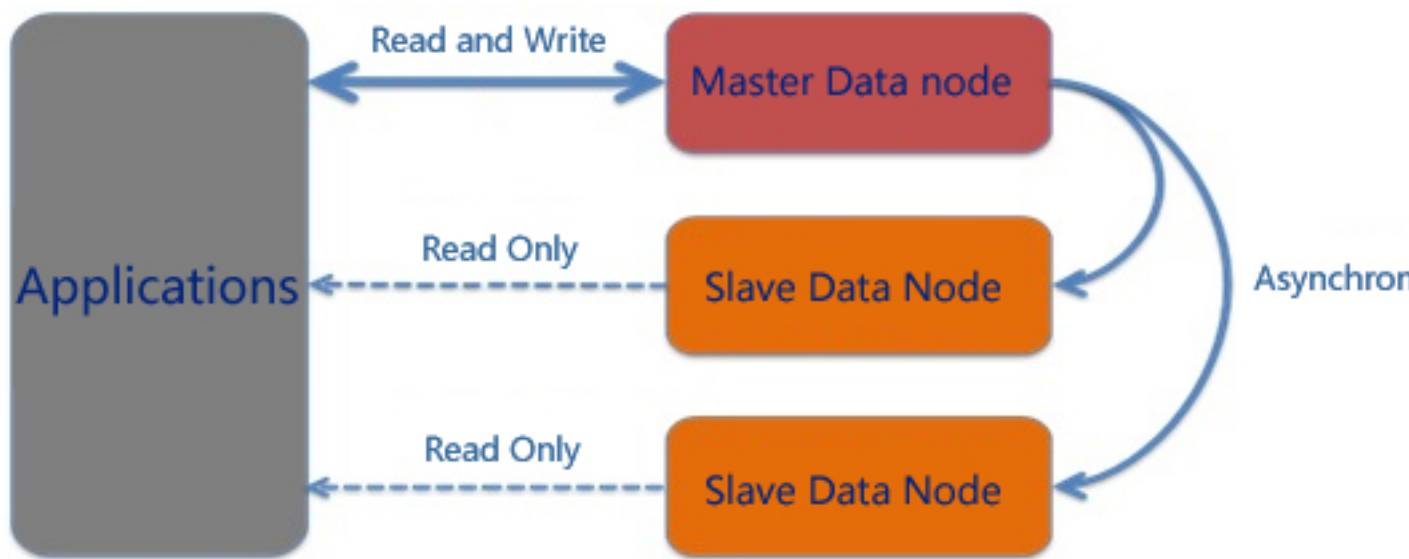
In a client terminal (or an application terminal), local or/and remote applications are linked to SequoiaDB client library. Local or/and remote applications communicate with catalog node under TCP/IP protocol.

Catalog node doesn't store any user data. It is only a node that receives requests and distributes them to target data nodes.

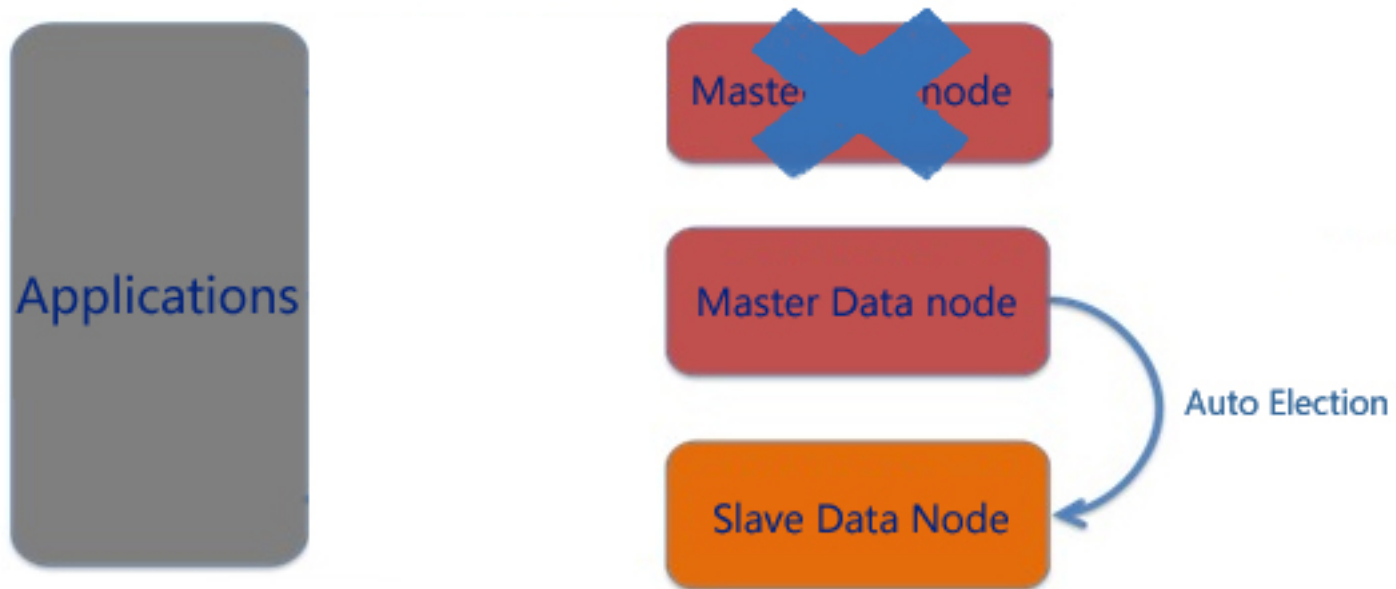
Catalog nodes store system metadata information. Coord nodes get the location of data on data nodes by communicating with catalog nodes. One or more catalog nodes can constitute a replset cluster.

Data nodes are used to store users' data information. One or more data nodes can constitute a replset. In a replset, data in all data nodes is eventually consistent. Data replset is also called shard. Different shards store different data.

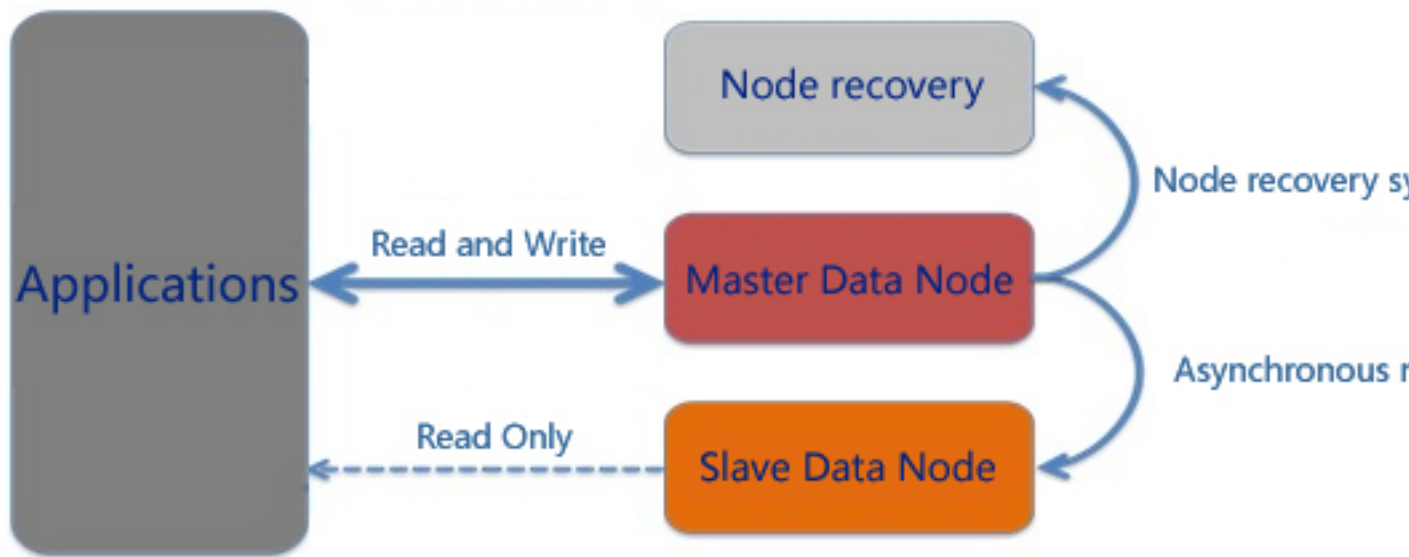
Every shard contains one or more data nodes. When there are several nodes in it, asynchronous replication is fulfilled. In a shard, there are master nodes and several slave nodes. Master nodes allow read and write operations. Slave nodes allow read operations.



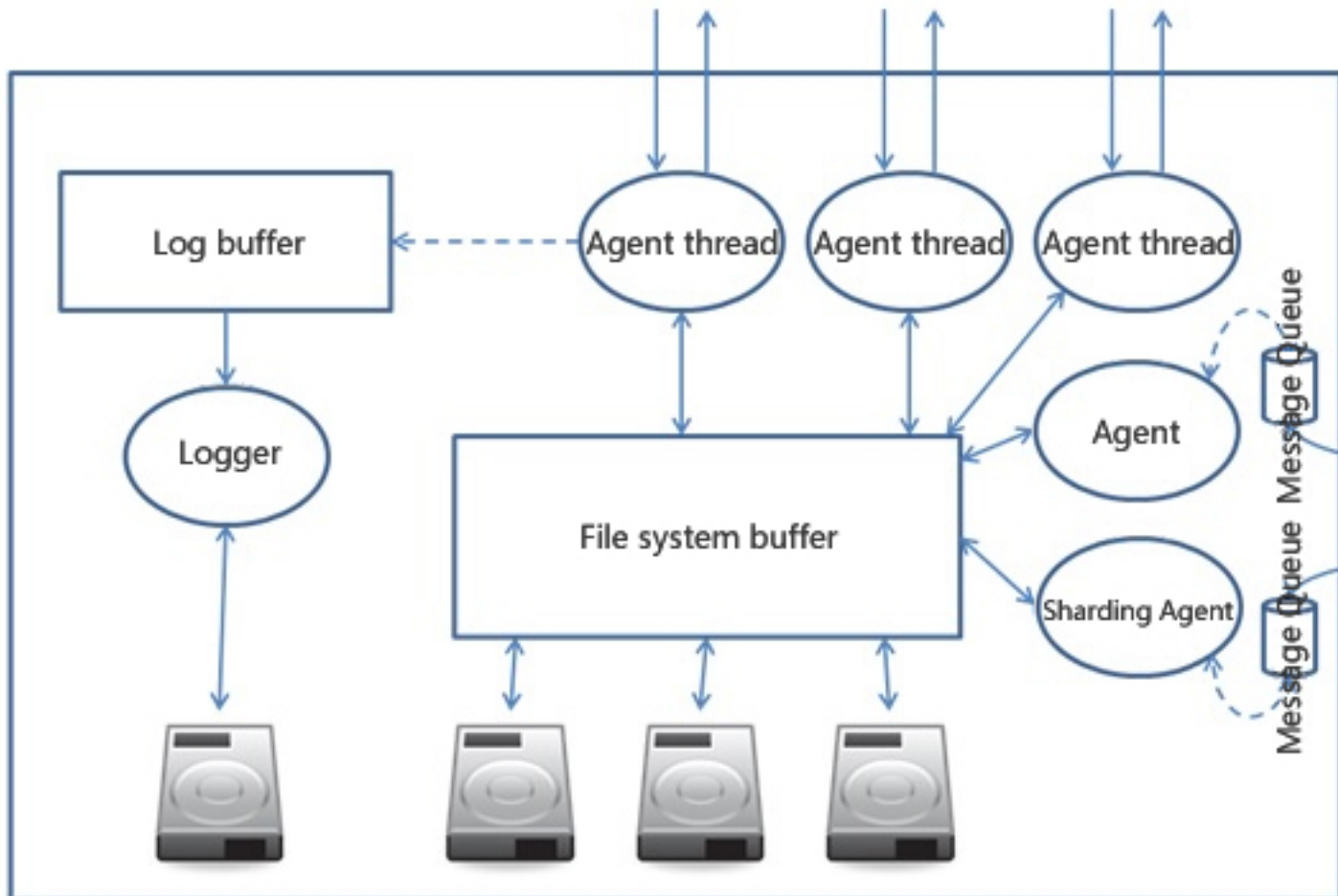
If slave nodes are off-line, master node can work well as usual. When master node goes down, slave nodes will automatically vote and elect a new master node to cope with write requests.



When broken-down nodes recover or new nodes join a shard, replication will be fulfilled between data nodes in order to guarantee the consistency of master node and slave nodes.



The architecture of a single node is as follow:



In data nodes, activities are controlled by EDU. Every node is a process in operating system. Every EDU is a thread in data nodes. Agent thread copes with user request from outside. Synchrononus agent thread copes with synchronized transaction within shards in cluster. Shard agent thread copes with synchronized transaction between shards in cluster.

All write operations are recorded in log cache. Log recorder will asynchronously write them into disk.

User data is written into file system cache pool by agent thread. Then operating system will asynchronously write them into underlying disk.

# Database Concept

Database concepts related content.

## Database

Database objects include document, collection, collection space and index.

## Document

### Concept

In SequoiaDB, document is in the format of JSON, also named as record. In database, JSON data is stored in the format of BSON after transformation. Generally, a document is consisted of one or more fields. Every field is consisted of 2 parts: key and value. This is a document containing 2 fields:

```
{ "name" : "Zhangsan", "gender" : "male" }
```



### Note:

BSON documents may have more than one field with the same name; however, most SequoiaDB interfaces represent SequoiaDB with a structure (e.g. a hash table) that does not support duplicate field names. If you need to manipulate documents that have more than one field with the same name, see your driver's documentation for more information.

Some documents created by internal SequoiaDB processes may have duplicate fields, but no SequoiaDB process will ever add duplicate keys to an existing user document.

### Field type

The key of a field is in the format of string, but value can be in the format of figure, string, nested JSON project, nested array, etc. SequoiaDB supports these types:

Type	Definition	Sample
Integer	Integer, range from -2147483648 to 2147483647	{ "key" : 123 }
Long	Integer, range from -9223372036854775808 to 9223372036854775807. If a value is not suitable to an Integer, it is automatically transformed into Long by SequoiaDB.	{ "key" : 3000000000 }
Float	Float, range from 1.7E-308 to 1.7E+308	{ "key" : 123.456 } or { "key" : 123e+50 }
String	String within double quotations	{ "key" : "value" }
Object ID (OID)	12-byte object ID	{ "key" : { "\$oid" : "123abcd00ef12358902300ef" } }
Bool	true or false	{ "key" : true } or { "key" : false }
Date	In the format of YYYY-MM-DD	{ "key" : { "\$date" : "2012-01-01" } }
Timeline	In the format of YYYY-MM-DD-HH.mm.ss.ffffff	{ "key" : { "\$timestamp" : "2012-01-01-13.14.26.124233" } }
Bindata	Base64 binary data	{ "key" : { "\$binary" : "aGVsbG8gd29ybGQ=", "\$type" : "1" } }
Regex	Regular expression	{ "key" : { "\$regex" : "^Zhang", "\$options" : "i" } }
Object	Nested JSON document object	{ "key" : { "subobj" : "value" } }

Type	Definition	Sample
Array	Nested array object	{ "key" : [ "abc", 0, "def" ] }
NULL	null	{ "key" : null }

### Field order

Fields are unordered. The order of fields may be changed in the process of data manipulation.

Within a nested object, fields are invoked in the format of "object.field". For example,

```
{ "name" : "zhangsan", "address" : { "street" : "shuilan street", "city" : "xx", "block" : "yy" } }
```

The field of "city" is invoked as "address.city".

### Others

- The max size of each document is 16MB
- Document should contains "\_id" field. If users do not offer it, syetem will automatically generate a field in the type of object ID.
- "\_id" is unique in a collection
- Field name in document should not begin with the character "\$".
- Field name in document should not contain ".".

### Array

#### Concept

Arrays in SequoiaDB, in the format of JSON object in documents, are generally named records.

#### Format

If there are multiple values in an array, users can store data in the format of data structure. Array begins with "[" and ends with "]". It contains 0 or multiple values.

```
{ Field Name : [ <Value1>, <Value2>, <Value3> ... ] }
```

#### Sample

Array can contain different data types as values. The index of elements start with 0. For example:

```
{ "key" : [ "hello", "world" ] }
```

The index of "hello" is 0. The subscript value of "world" is 1. The values in an array are in order. The order of values is unchangable when operation is implemented in an array.

Element in array is invoked as "field name.index". For exmaple, in the sample above, "world" is invoked as "key.1".

### Object ID

#### Concept

Object ID is a 12-bit BSON object. It contains:

- 4-byte timeline measured in seconds
- 3-byte system (physical machine) identity
- 2-byte process ID
- 3-byte sequence number that starts with a random number

4-byte timeline	3-byte system identity	2-byte process ID	3-byte sequence number

The object ID can identify each process of each system in cluster environment with 16777216 different values, so it is regarded as a globally unique value in cluster environment.

In SequoiaDB, each file in a collection stores at least an "id" field, which is unique within the collection.

#### Format

The format of an object ID is:

```
{ "$oid" : "<24-byte hexademical string>" }
```

#### Sample

Object ID is displayed as:

```
{ "key" : { "$oid" : "5156c192f970aed30c020000" } }
```

#### Date

#### Concept

In SequoiaDB, date is in the format of YYYY-MM-DD. It is transformed into a 4-byte integer when it is stored in the database.

#### Form

The format of date is :

```
{ "$date" : "<YYYY-MM-DD>" }
```

#### Sample

For example:

```
{ "createTime" : { "$date" : "2012-05-12" } }
```

#### Timeline

#### Concept

In SequoiaDB, timeline is in the format of YYYY-MM-DD-HH.mm.ss.ffffff. It is transformed into an 8-byte integer when it is stored in the database.

#### Format

The format of timeline is :

```
{ "$timestamp" : "<YYYY-MM-DD-HH. mm. ss. fffffff>" }
```

#### Sample

For example :

```
{ "createTime" : { "$timestamp" : "2012-05-12-13.15.21.241523" } }
```

#### Bindata

#### Concept

In SequoiaDB, data is read in the format of JSON object. So users should encode bindata with Base64, and send it to database in the format of string.



## Format

The format of bindat is as follow

```
{ "$binary" : "<data>", "$type" : "<type>" }
```

In this format, "data" should be encoded with Base64. "Type" is a decimal value between 0 and 255. Users can choose type values in this range to identify types in applications.

Base64 is an universal format of transforming data, which is mainly transforming bindata into byte stream in the format of ASCII string. Generally, data becomes longer after transformation.

In order to save storage space, in SequoiaDB, Base64 data is transformed into original data before being stored in database. When users request to read data, it is transformed and provided in the format of Base64 again.

## Sample

For example, string "hello world" is encoded with Base64 into "aGVsbG8gd29ybGQ=". JSON data containing "hello world" bindata, with type of "1" is:

```
{ "key" : { "$binary" : "aGVsbG8gd29ybGQ=", "$type" : 1 } }
```

## Regex

### Concept

SequoiaDB can search for user data with regex.

## Format

The format of input is :

```
{ "$regex" : "regex", "$options" : "options" }
```

Regular expression is a regex string. Options can be:

Options	Description
i	Case-insensitive.
m	Allow mutiple match; when the parameter is set, "^" and "&" respectively matches characters before and after a line break.
x	Ignore blank characters matched by regex. When blank character is needed, it should be prefixed with "W", the escape character.
s	Allow to match line break with ".".

When using options, users can choose several options at a time.

## Sample

Users can match string that is case-insensitive and begins with the character "W" with:

```
{ "key" : { "$regex" : "^W", "$options" : "i" } }
```

Please refer to [Perl Regex Manual](#) for more about rules of regex.

## Collection

### Concept

Collection a logical object where documents are stored. Every document belongs to one and only one collection.

Collection consists of "<collection space name>.<collection name>". The length of a collection name is at most 127 bytes, encoded with UTF-8. A collection contains zero or more documents. (The max amount of documents depends on the size of collection space.)

In sharding enviroment, apart from name, every collection contains three attributes.

field name	description
Sharding Key	Specified sharding key. In collections, the field specified by sharding key is used as sharding information of document. Every document is put in its corresponding sharding.
ShardingType	Specified set partition type:range partitioning (Range) or hash partitioning (Hash).
ReplSize	It specifies the default replicaion size in the collection. If its value is less than 1, write request will return when one copy is written successfully. If its value is more than 1, write request will return when the amount of successfully written copies euqals at least the value of ReplSize.
Compressed	Specified the value of "Compressed" means whether stored data in compressed form, it has two values:true and false, false is in default.

## Collection Space

### Concept

Collection Space is physical object where collections store. Every collection belongs to onre and only one collection space.

The length of collection space name is at most 127 bytes, encoded with UTF-8. A collection space contains at most 4096 collections. Every data node contains at most 4096 collection spaces.

Every collection space has a corresponding file on the data node. The format of file name is <collection space name>.1".

### Page

A file is departed in to several fixed-size pages by space collection. When creating a collection space, users can specify the size of a page in it, which is unchangeable.

In every sharding node, a collection space can visit at most 16777216 different pages. According to pages of different sizes, the max size of a single sharding is:

Size of page ( byte )	Collection space max capacity ( GB )
4096	512
8192	1024
16384	2048
32768	4096
65536	8192

### Block

A block is consisted of one or more pages. Every collection in collection spaces is consisted of zero or more blocks. The size of block is changeable according to the length of data. Every document is stored in only one block.

Blocks in a collection space are stored as follow:



In this chart, there are 3 collection in a collection space, represented by 3 colors. Data of every collection is stored in its corresponding page. Data block is consisted of one or more sequential data pages. A document cannot be divided and stored in different data blocks.

## Database Server

### Concept

Database server provides software services for the purpose of managing information securely and efficiently.

A database server is a computer that has installed SequoiaDB database engine. SequoiaDB engine is the basic unit of access to data. In distributed architecture, every database is regarded as a node. These nodes share nothing.

In a computer, every SequoiaDB engine has a database path. All the collection spaces of the database is located in the directory.

Database path contains one or more collection spaces. In a data node or a coord node, there should be at least one SYSTEMP system temporary collection space. In a catalog node, there should be at least one SYSTEMP system temporary collection space, SYSAUTH system permissions collection space, SYS PROCEDURES system stored procedure collections space and one SYSCAT system catalog collection space.

A database engine contains 4096 collection spaces at most.

## Index

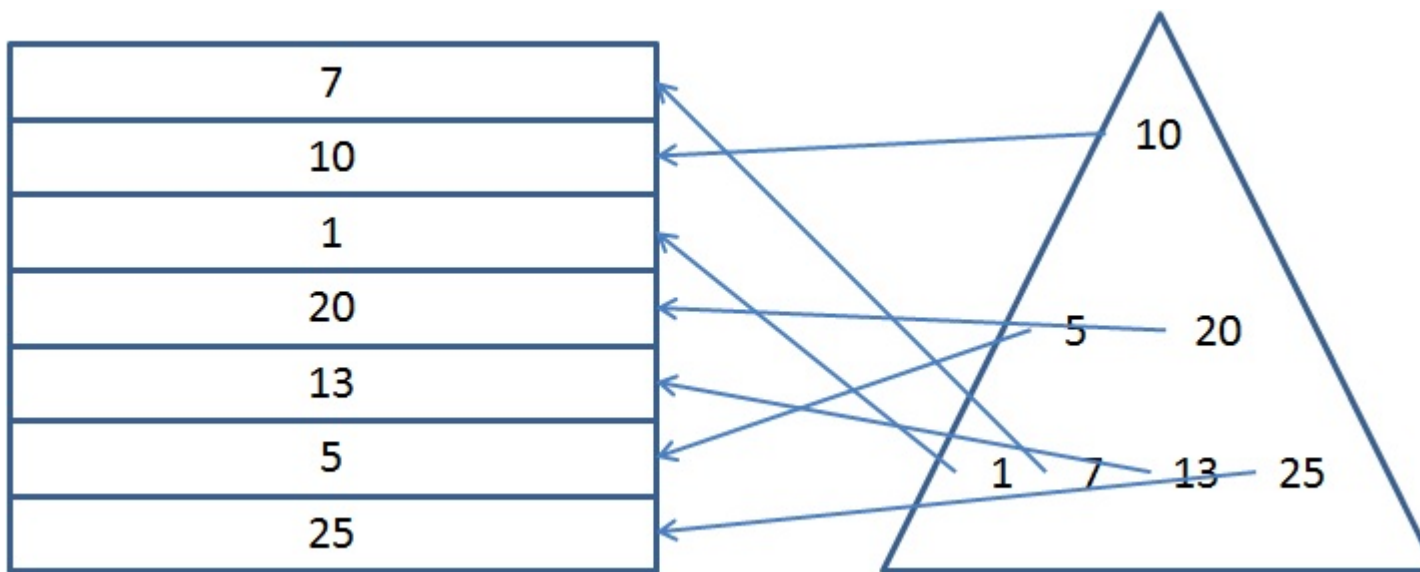
### Concept

In SequoiaDB database, indexes are a kind of special objects. Instead of being used to store user data, indexes, a kind of special metadata, optimize speed and performance in finding relevant documents for a search query.

Every index is put in a collection. A collection contains 64 indexes at most.

Indexing is regarded as an approach to sort data based on one or more fields, and rapidly get results which match users' query statements from it as soon as queries are submitted. In SequoiaDB, indexes are stored in structures called B-trees.

A typical index structure is as follow :



In this figure, the data is on the left, the indexes are on the right in the triangle. Indexes are sorted in the structure of trees in ascending order. every document will have a unique index value.

Through tree traversal, the position of specific data can be rapidly found.

SequoiaDB can create indexes to any type of data. An index contains:

Attribute	Description
name	Index name. Index name is unique in a collection.
key	Index key, as a JSON object, contains one or multiple indexes field and direction field. If direction field is 1 or -1, the order is ascending or descending.
unique	"Unique", a optional parameter, shows that whether an index is unique. The default value of it is "false". When its value is "true", the index is unique. As for unique index, there should not be repetitive records in a collection.
enforced	"Enforced" shows that whether an index is enforced to be unique. The default value is "false". If its value is "true", with "unique" set "true", there is no more than one null index key.

In SequoiaDB, every collection contains an enforced unique index named "\$id". The index contains a index key of "\_id".

An index named "\$shard" is automatically is generated when sharding collection is created. The index key of it is the sharding key specified by users.



Note:

In sharding collection, all unique indexes should contain all the fields specified by collection sharding key.

In sharding collection, "\$id" index merely ensure the uniqueness of records within a single node. If users plan to create a globally unique field, they have to create an extra unique field, which contains all the fields specified by collection sharding key.

## Format

The format of index contains two fields: "name" and "key". The type of "name" is string. The type of "key" is JSON object.

```
{ "name" : "<index name>", "key" : "{ "<index field1>" : <direction> [, "<index field2>" :
<direction> ...] }",
  [ "unique" : <true|false> ], [ "enforced" : <true|false> ] }
```

The object of "key" contains at least one field. The field name of it is the field name that user search for. Its value is "1" or "-1". "1" represents ascending sequence of data. "-1" represents descending sequence of data.

## Sample

- Non-unique index, index name: "employee\_id\_key", index field is forward "employee\_id".

```
{ "name" : "employee_id_key", "key" : { "employee_id" : 1 } }
```

- Unique index, index name is "record\_id\_index", index field is forward "product\_key" and reverse "record\_key".

```
{ "name" : "record_id_index", "key" : { "product_key" : 1, "record_key" : -1 }, "unique" :
true }
```

In this index, there are no two records with the same "product\_key" and the same "record\_key". (But if either of them is the same, it is still regarded as an unique index).

- Enforced unique index. The index name is "test index". The index field is ascending "test case name".

```
{ "name" : "test index", "key" : { "test case name" : 1 }, "unique" : true, "enforced" :
true }
```

In an enforced unique index, all records should follow the rule of unique index. There should not be more than one record that contains a null value of "test case name" field.

## Aggregate

Aggregation framework provides for the collection of raw data records for statistical computing. By using the aggregation framework, users can extract data directly from the collection of statistical records and obtain the required result. Aggregation framework provides a collection of user interface similar to the query operation, unlike the aggregation framework also provides a set of functions and operators to process the query results.

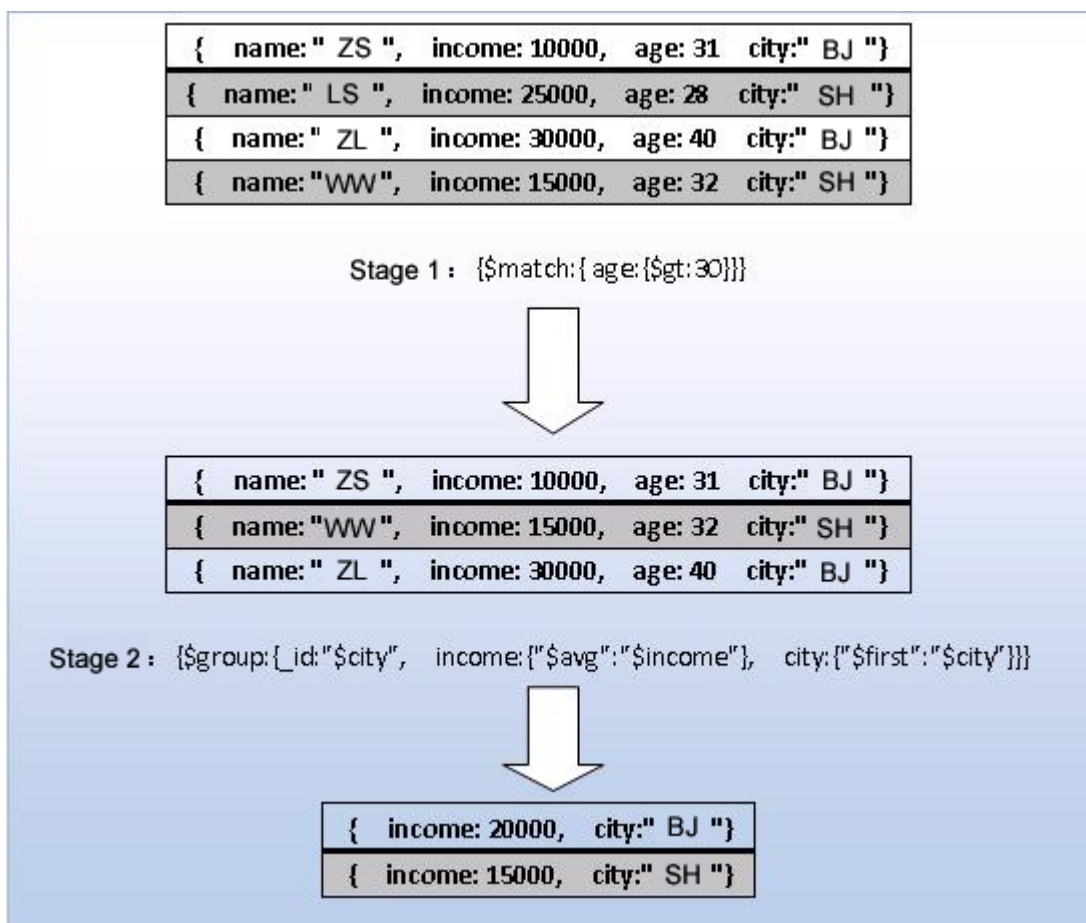
[More aggregation operator operation, please click here.](#)

### aggregate()

The following is an example aggregation operations:

```
db.space.cl.aggregate(
  {$match: { age: {$gt: 30}}},
  {$group: { _id: "$city", income: { "$avg": "$income" }, city: { "$first": "$city" } } })
```

Aggregate operation on case contains two sub-operations, where "\$match" sub-operation in the collection of data records older than 30 screened out; "\$group" operations from the filtered data records are grouped according to the city, calculated for each city the per capita income. Through the example aggregation operation will be over 30 years old each city's per capita income.



## Transaction

Transaction is consisted of a series of actions by logical unit of work. In the same session (or connection), only allow exist one transaction in the same time. For another word, when the users create a transaction in a session, before the end of the transaction, the users can't create new transaction.

Transaction performed as a complete unit of work. The operations in transaction execute either all succeed or all fail. In SequoiaDB, operations of transaction can only be insert, modify and delete data, others will not be included in transaction. It is said that non-transactional operations will not perform rollback when execute transaction rollback action. If a collection or collectionspace have data relate to transaction operation, then the collection or collectionspace is not allowed to drop.

In default, transaction function is close, if users want to open it, need to configure param in the configuration file of nodes: `transaction=TRUE`; when create node, add json type of param: `{"transaction": "YES"}` or `{"transaction": true}`.

## Eventually Consistent

In SequoiaDB, to improve data reliability and data read/write Splitting, we adopt "eventually consistent strategy" among ReplicaSets. Maybe, the data is not latest in a moment, but ultimately the same.

### Glossary

W : number of write-replicas

R : number of read-replicas

N : number of replicas

In SequoiaDB, the value of "R" is default set to "1", and not configurable.

By default, The master node in data ReplicaSets will return immediately after processing a written request, that is  $W=1$ . Data syn will be done asyn in the background ([Sys-log](#)) and achieved eventually consistent. At this moment, the data for external read request may not be the latest. In the case of less demanding data consistency, this way will provide a optimal write-performance.

As the time of [creating collection](#), you can set the value of "W" by "ReplSize" param.

- By default,  $W=1$ .
- If the value of ReplSize is equal "0", then the number of W will be changed based on the number of nodes in current ReplicaSets. That is, there have three nodes in a ReplicaSet at beginning, then W is equal 3; if adding a node, then W change to "4".
- if specify the number of W by manually, It can not exceed the number of nodes in current ReplicaSet.

Increasing the value of W can effectively improve the consistency and reliability of data. When the value of W is equal to the number of nodes in a ReplicaSet and write-request is done successfully, then the data for read-request are always the latest. But this will reduce the write-performance. For attention, although, the value of W can be set to equal to the number of nodes, it not mean that the data in SequoiaDB have strong consistency. When a Replica write failed (such as the disk is full), it may exist one or more data versions in a ReplicaSet, this time, you may read the latest data, and may also the older. When the failed replica get right, it will syn the latest data from the master node, and achieve eventually consistent.

## Examples

**Replica Group Examples** Copy each data node in the group is a complete data stored in the replication group, and therefore also called copy group copy for each node instance. Examples of replication group can be divided into "main", "prepared" or "0~7" logo based on the node position in the replication group.

**Database instance** All instances of copy group copy group together constitute the same position database instance, the database instance can also be divided into "main", "prepared" or "0~7" logo.

## Read/Write Splitting

### Write-request

All of write-requests will only be sent to the master node, no master, no write-request.

### Read-request

It will based on the session (or the connection) randomly select one node in a replicaset for read-request (external transparent). If the last query operation (including query, fetch) returned success, then the next will not re-elect node, otherwise, the next query operation have to select node. If there have no available node, return failure. In one query operation, will not re-elect.

## Background Task

Background task is SequoiaDB a particular type of task, generally for a specific user actions into the background asynchronously. In the snapshot, the background task type (Type) as "Task".

Background task list type:

Task name	Description
Restore	Database recovery task, according to the log file for database rollback recovery.
Job[PageCleaner]	Dirty pages cleanup tasks will not be written for asynchronous disk brush dirty pages to disk. You can use the "-numpagecleaners" control the number of dirty pages clear mission, the default is 1.
Job[Prefetch]	Prefetch tasks while waiting for the next client receives an operation request to perform the next operation of the user may occur in the

Task name	Description
	background. You can use "-maxprefpool" to control the maximum number of prefetch task.
CreateIndex	Indexing tasks for background indexing, multi-node for heavy equipment operation log shots indexing node.
DropIndex	Delete indexing task, delete the index for the background, used heavy equipment shots node node to delete the index operation log.
CleanUp	Data cleaning tasks, and more after the cut points for data, delete the source data node segmentation data.
Job[ExtendSegment]	File extension collection space mission, when a collection of space for free after less than a certain threshold data pages form the background to start a collection of asynchronous expansion space.

## Cluster

---

SequoiaDB cluster is a mode designed to improve the efficiency of data request by achieving parallel computing multiplied database servers

Using SequoiaDB cluster , a user can get:

### High-qualified Data Visit

With parallel computing and executing distributed tasks on more than one data server, the system saves resource consumption on every nodes, gets higher throughput and improves the quality of data visit.

### 24x7 High Usability

Data replication in replsets guarantees data usability, so if any server doesn't work accidentally, data can be visited as well.

### Horizon dilation ability of database

Database can contain more data by adding more replsets. Database can cope with more requests by adding more nodes in a replset. The more database nodes, the higher global throughput.

## Mode

### Concept

Mode means the approach of starting SequoiaDB service. Generally, it is independent mode or cluster mode.

### Independent Mode

Independent mode is the briefest mode to start SequoiaDB. It merely needs to start a data node of **independent mode** before having data service.

Under independent mode, SequoiaDB Database is an independent process which doesn't need to communicate with other clients. All the data is stored in data nodes.

In a database started under independent mode, it is not allowed to split space or replicate data. So it is not recommended to use independent mode if high data security is required.

In a database start under independent mode, there is no catalogue information.

It is recommended to use independent mode in development environment in order to reduce requirement of hard resource.

### Cluster Mode

Cluster mode is the standard mode to start SequoiaDB, which requires 3 nodes.



Under cluster mode, SequoiaDB needs 3 kinds of roles:

- data node
- catalogue node
- coordinating node

The minimum configuration under cluster mode contains at least one of each kind of roles, which is essential to build a complete cluster mode.

Under cluster mode, client and application can connect to coordinating node directly. Other kind of nodes are invisible to them.

Applications care nothing about the location of data in data nodes. Coord node can analyze request that it receives and automatically send it to relevant data nodes.

Under cluster mode, data set is unvisable to each other. In order to achieve data consistence, the data replication among nodes within a data set is asynchronous.

## Node

### Concept

In cluster mode of SequoiaDB, there are 2 kinds of nodes: physical nodes and logical nodes

#### Logical Node

Logical node is an independent service, representing the most basic database service logical unit.

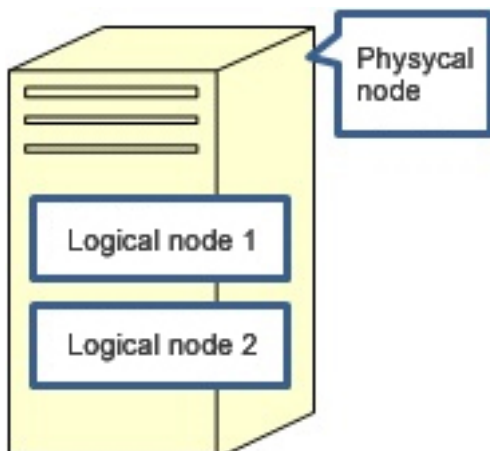
In Windows OS and Unix OS, a logical node is a sequoiaDB process. There can be mutiple logical nodes running on a compute.

#### Physical Node

Physical node is a server which runs an OS, representing the most basic database service physical unit.

In virtual environment, a physical node is a vortual machine.

Multiple logical nodes can run on a physical nodes. Gnerally, each logical node on the same physical node listens to different ports, receives request from their corresponding ports, and returns results.



## Catalog Node

### Concept

Catalog node is a kind of logical node, which stores the metadata of a database system instead of data of other users.

Catalog node includes 3 space collections:

- SYSCAT system catalog contains 4 system collections:

collection name	descripton
SYSCOLLECTIONS	store user collection information in the cluster
SYSCOLLECTIONSPACES	store user collection space information in the cluster
SYSNODES	store information of logic nodes and replsets in the cluster
SYSTASKS	store information of background-runnnig tasks in the cluster

- SYSTEMP system temporary collection space is used to create at most 4096 temporary collections in order to sequence data when users visit the database.
- SYSAUTH system validation collection space contains a user colleciton, which stores information of all the users in current system

collection name	descripton
SYSUSRS	stores all the user information in the cluster

- SYSPROCEDURES collection space system stored procedure that contains a collection of all of the stored procedures for storing function information

collection name	descripton
SYSPROCEDURES	save all the information stored procedure function

Except for catalog nodes, all the other nodes in the cluster don't store any global metadata information in the disk memory. When users want to visit data on other nodes, the system will find collection information from local cache. If it is not found, the system will search for it in catalog nodes.

Catalog nodes contacts with other nodes mainly through catalog service port(catalogname parameter).

### SYSCOLLECTIONS Collection

### Collection Space

#### SYSCAT

### Concept

SYSTASKS collection contains information of all user collections. Information of a user collection is saved in a corresponding file.

Every file contains the following fields:

Field Name	Type	Description
Name	String	Full name of a collection. The form is <collection space>.<collection name>.
Version	Integer	Version number of the collection, starts with 1. It increases by 1 after any metadata alteration on the collection.
ReplSize	Integer	The minimum replica group, to ensure that any write operation must be copied to at least return after a specified number of nodes success.

Field Name	Type	Description
ShardingKey	Object	ShardingKey is in sharding collections. It contains one or more fields. The field name is its sharding field name. The value is 1 or -1, which represents forward or reserve sequencing.
ShardingType	String	Partition types: the presence in the partition collection. Partition types: range partitioning(Range) and hash partitioning(Hash) two.
Partition	Integer	The partition size of the hash value must be a power of 2.
CataInfo	Array	Information of logical nodes of a collection. In a single sharding collection, the array contains only one element, which represents the replset of the collection. In a multiple sharding collection, the array contains one or more elements, which represent the replset which covers all the ranges of values in the collection. Every range of values includes two value: LowBound and UpBound, which respectively represent floor and ceiling. The interval is left-close and right-open.

### Sample

A typical single sharding collection is as follow:

```
{ "Name" : "test.foo", "Version" : 1, "CataInfo" : [ { "GroupID" : 1000 } ] }
```

A typical multiple sharding collection is as follow:

```
{ "Name" : "foo.test",
  "Version" : 1,
  "ShardingKey" : { "Field1" : 1, "Field2" : -1 },
  "ShardingType" : "range",
  "ReplSize": 3,
  "CataInfo" : [
    { "GroupID" : 1000,
      "LowBound" : { "" : MinKey, "" : MaxKey },
      "UpBound" : { "" : MaxKey, "" : MinKey } } ]
}
```

### SYSCOLLECTIONSPACES Collection

#### Collection Space

#### SYSCAT

#### Concept

SYSCOLLECTIONSPACES collection contains information of all user collection spaces. Information of a user collection space is saved in a corresponding file.

Every file contains the following fields:

Field Name	Type	Description
Name	String	Collection space name.
Collection	Array	All the collection names of the collection space. Each collection is a JSON object, which

Field Name	Type	Description
		contains the field "Name" and the name of the corresponding collection.
Group	Array	The replset ID of the collection space.
PageSize	Integer	The size of a data page in the collection space.

### Sample

A typical collection space, stored in a replset, that contains one collection is as follow:

```
{ "Collection" : [ { "Name" : "foo" } ],
  "Group" : [ { "GroupID" : 1000 } ],
  "Name" : "test",
  "PageSize" : 4096 }
```

### SYSNODES Collection

#### Collection Space

#### SYSCAT

### Concept

SYSNODES collection contains information of all the nodes in the cluster and of replset. Information of a replset is saved in a corresponding file.

Every file contains fields as follow:

Field Name	Type	Description
GroupName	String	Replset name.
GroupID	Integer	Replset ID. It is unique in the cluster.
PrimaryNode	Integer	The ID of the master node in a replset.
Role	Integer	The role of a replset. It can be: <ul style="list-style-type: none"> <li>0: data node</li> <li>2: cataloge node</li> </ul>
Status	Integer	<ul style="list-style-type: none"> <li>1: activate replset</li> <li>0: inactive replset</li> <li>not exists: inactive replset</li> </ul>
Version	Integer	Version number. It starts with 1. It increases by 1 after any execution on the replset.
Group	Array	It contains information of nodes in a replset,as it is shown as follow:

If there are more than one node in a replset, each node is store in Group as an object. Each object contains:

Field Nmae	Type	Description
HostName	String	The hostname which a node belongs to, should absolutely matches the output of the instruction, "hostname", that runs on the operating system in which the node exists.
dbpath	String	Database path, is the absolute path of a physical node that the node belongs to.
NodeID	Integer	Node ID,is unique in the cluster.

Field Name	Type	Description
Service	Array	<p>Server name. Every logical node corresponds to 4 services. Every service contains its type and server name (it may be port number of server name in service files). The types are as follow:</p> <ul style="list-style-type: none"> <li>• 0 : direct service, corresponds to database argument svcname</li> <li>• 1 : replicate service, corresponds to database argument replname</li> <li>• 2 : replset service, corresponds to database argument shardname</li> <li>• 3 : catalog service, corresponds to database argument catalogname</li> </ul>

**Note:**

The group name of a catalog replset should be "SYSCatalogGroup". The ID of a catalog replset should be 1.

Data replset ID starts from 1000.

Data node ID starts from 1000.

**Sample**

A typical catalog replset that contains only one node is:

```
{ "Group" : [
  { "NodeID" : 2,
    "HostName" : "vmsvr1-rhel-x64",
    "Service" : [
      { "Type" : 3, "Name" : "11803" },
      { "Type" : 1, "Name" : "11801" },
      { "Type" : 2, "Name" : "11802" },
      { "Type" : 0, "Name" : "11800" } ],
    "dbpath" : "/home/sequoiadb/sequoiadb/catalog"
  } ],
  "GroupID" : 1,
  "GroupName" : "SYSCatalogGroup",
  "PrimaryNode" : 2,
  "Role" : 2,
  "Version" : 1 } }
```

A typical data replset that contains only one node is:

```
{ "Group" : [
  { "dbpath" : "/home/sequoiadb/sequoiadb/data3",
    "HostName" : "vmsvr1-rhel-x64",
    "Service" : [
      { "Type" : 0, "Name" : "11820" },
      { "Type" : 1, "Name" : "11821" },
      { "Type" : 2, "Name" : "11822" },
      { "Type" : 3, "Name" : "11823" } ],
    "NodeID" : 1001 } ],
  "GroupID" : 1001,
  "GroupName" : "foo1",
  "PrimaryNode" : 1001,
  "Role" : 0,
  "Status" : 1,
  "Version" : 1 } }
```

## SYSTASKS Collection

## Collection Space

## SYSCAT

## Concept

SYSTASKS collection contains information of all the running background tasks. Information of a task is saved in a corresponding file.

Every file contains the following fields:

Field Name	Type	Description
JobType	Integer	Task type, respectively represents: <ul style="list-style-type: none"> <li>0: data split</li> </ul>
Status	Integer	Task status, respectively represents: <ul style="list-style-type: none"> <li>0: preparing</li> <li>1: running</li> <li>2: paused</li> <li>3: cancel</li> <li>4: change metadata</li> <li>9: completed</li> </ul>
CollectionSpace	String	Collection space same
Collection	String	Collection name

## Data Split

Regarding data split, every file contains the following fields:

Field Name	String	Description
SourceName	String	Source replset name
TargetName	String	Target replset name
SourceID	Integer	Source replset ID
TargetID	Integer	Target replset ID
SplitValue	Object	Data split key

## SYSUSRS Collection

## Collection Space

## SYSAUTH

## Concept

SYSUSRS collection contains information of all registered users in the cluster. Information of a user is saved in a corresponding file.

Every file includes :

Name	Type	Description
User	String	User name.
Password	String	MD5 hash value of password.



Note:

If the collection is empty, it won't provide any connection with identification validation .

## STOREPROCEDURES Collection

### Collection Space

### STOREPROCEDURES

### Concept

STOREPROCEDURES collection contains all of the stored procedure function, each function is saved as a document, each document contains the following fields:

Field Name	Type	Description
name	String	Function name.
func	String	Function body.
funcType	Integer	Function type. <ul style="list-style-type: none"> <li>0 : Representative javascript function.</li> </ul> No other type of.

### Example

A simple stored procedure functions as follows.

```
{
  "_id" : { "$oid" : "5257b115925c31dd16ec4e4a" },
  "name" : "fun",
  "func" : "function fun(num) {
    if (num == 1) {
      return 1;
    } else {
      return fun(num - 1) * num;
    }
  }",
  "funcType" : 0 }
```

### Coord Node

### Concept

Coord node is a kind of logical node, which stores information except user data information.

Coord node is the coordinator of data requests, which merely send requests to relative data nodes rather than takes part in data execution.

Generally, the process of executing coord node is as follow:

- get a request
- analyze the request
- find cooresponding information of the request in local cache
- if not found, find it in catalogue nodes
- send the request to corresponding data node
- get result from data node
- collect results and send them to client

Coord node contacts with other nodes mainly through catalog service port(shardname parameter).

## Data Node

### Concept

Data node is a kind of logical node, which stores user data information.

Data node doesn't contain special catalogue information collection, so before the first visit to a collection, it is essential to get metadata information from catalog information collection.

Under independent mode, a data node ,as an independent service provider, is able to communicate with application and client without visiting any catalogue information collection.

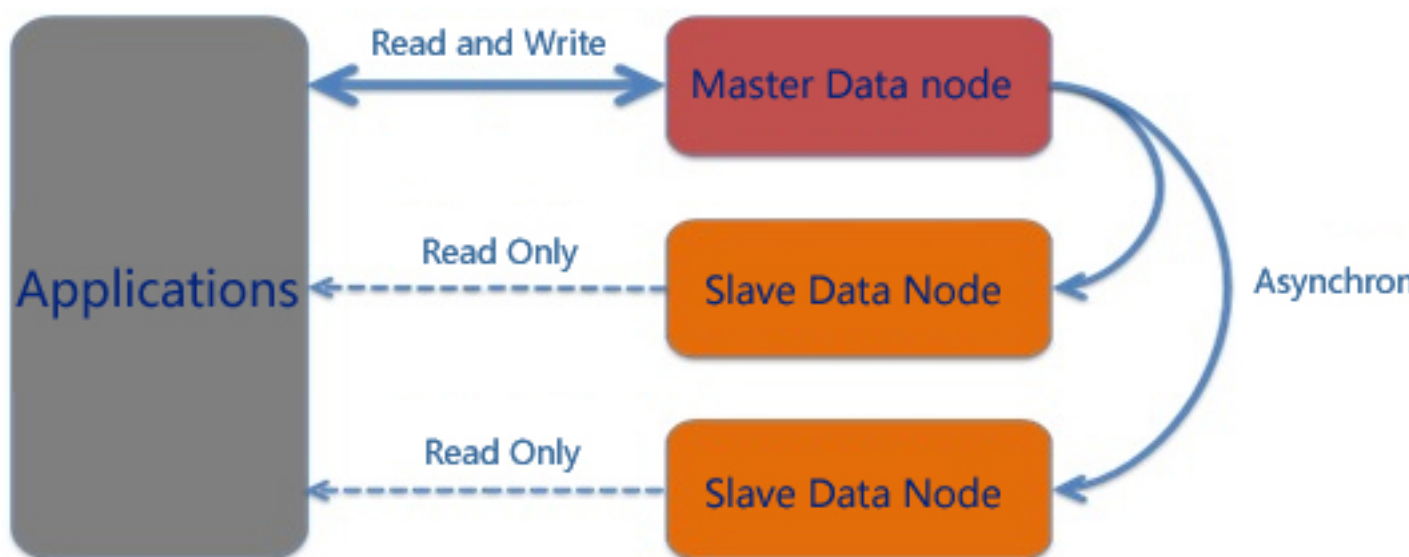
## ReplicaGroup

### Concept

ReplicaGroup is also named as replication group. A replset may contain one or more data nodes (or catalog node).The data replication between nodes is asynchronous log file replication,which ensures data consistence.

All the nodes in a replset contacts with each other through replication service port (replname parament), and regularly sent heartbeat package to each other to validate their states.

The structure of replset is as follow:



Each node in a replset has two statuses:

Master node

Master node is also called main node. Master node can be written and read. All the written fata is recorded in log file aynchronously. Log information in log files is written into nodes asynchronously.

Slave node

Slave node is also called slave node. Slave node can only be read. All the data written into master node is asynchronously written into slave node. So the data in main node and inferior may be inconsistent, but replication can guarantee data consistence.

Replset achieves high usability through data replication, read/write splitting and vote.

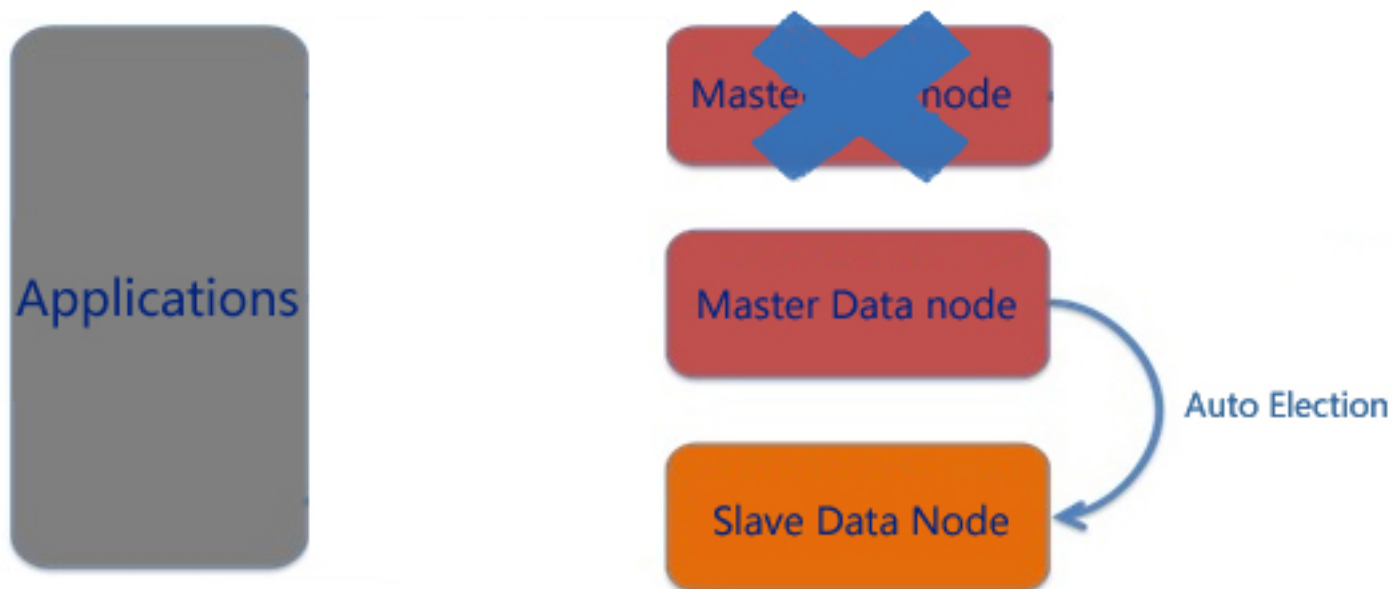


## Vote

### Concept

Vote guarantees that there is a master node in a replset at any time. When the master node in a replset goes down, other nodes will automatically vote for another master node among them. In this way, reading and writing can be executed as usual when the master node goes down.

The core of vote is to supervise the statuses of nodes. Each node in a replset regularly sends its status to other nodes. So when the master node goes down, all the slave nodes will vote, and the node that currently matches the former master node will be elected to be the new main node.



The precondition of a successful vote is that more than half of nodes take part in the vote, or the vote will be canceled to avoid the problem of double-activation (two main nodes exist at the same time).

If there are less than half of nodes in the replset, the current master node will automatically become a slave node. Meanwhile, all the user connections to the current node will be disconnected.

When a new node joins a replset, or a broken-down node joins a replset again, [full sync](#) will be fulfilled.

## Replicate

### Concept

Data replication is synchronously executed between two nodes in a replset.

Any insertion, deletion or alteration in data nodes or catalog nodes will be recorded in a log file. SequoiaDB puts the log file in a log cache, and then asynchronously stores it in disk memory.

Every data replication is executed between nodes:

Source Node

It is a kind of node containing new data. Master is not always the source node in the process of replication. Slave node can also be a source node.

Target Node

It is a node to copy data for a request.

In the process of replication between 2 nodes, the target node chooses the closest node to it, then sends a replication request to it. After the source node gets the request, it will package all the log records after the synchronous timeline provided by the source node and send them to the target node.

The target node copes with all the manipulation in log files after reviewing data package.

There are 2 statuses in the replication between nodes :

- PEER : When the log files requested by a target node is still stored in the log cache of source node, the status of the two nodes is PEER.
- Remote Catchup: When the log files requested by the target node is not stored in the log cache of source node and is still stored in the log file of source node, the status of the 2 nodes is "Remote Catchup".

If the log files requested by the target node is not stored in the log file of source node, the target node will enter the status of **full sync**

Regarding the status of PEER, the system can directly read data from RAM in response to sync request. So when target node choose source node, it attempts to choose the closest node according to current log files. In this way, it can save time by reading data from RAM in most cases.

### Full Sync

#### Concept

When a new node joins a replset or a broken-down node returns to a replset, it is essential to implement full sync to guarantee the consistence of data between the new node and other nodes.

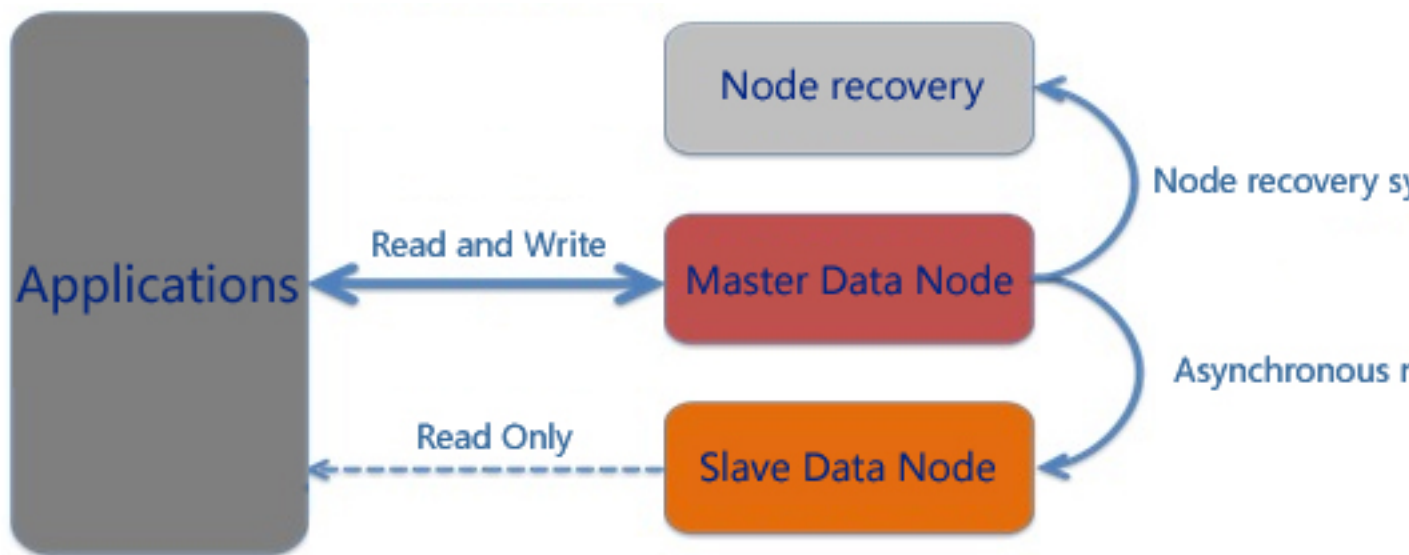
In the process of full sync, two nodes take part in it:

Source node

Source node contains valid data. Master node is not always source node in full sync. Any slave node synchronous with the master node can be a source node in full sync as well.

Target node

Target node is a new node that joins a replset or a broken-down node that returns to a replset. In full sync, original data on target node is discarded.



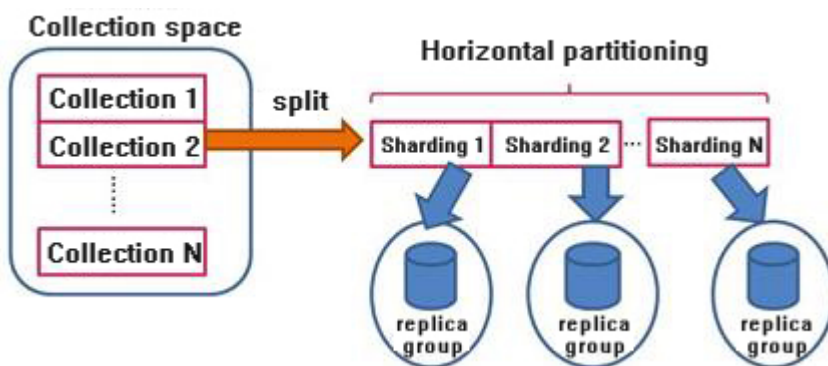
In the process of full sync, a target node will regularly request data from a source node. The source node packs data into big data block and sends it to the target node. When the target node receives all the data in the block, it will request new data block from the source node.

In order to guarantee accessible writing on source node, if any data page sent to the target node is alerted again, it will be updated to the target node. In this way, new data is not discarded in this process.

## Sharding

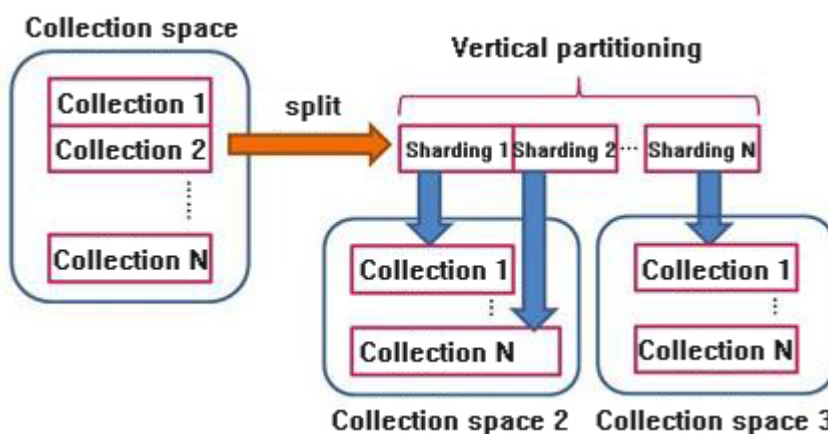
**Horizontal Partitioning** Horizontal partition database partition called double or horizontal partition.

In SequoiaDB cluster environment, the user can be cut in a collection of data into multiple replication groups in order to achieve the purpose parallel computing, the data segmentation called horizontal partitioning. Horizontal partitioning is based on certain conditions all tuples divided into a number of global relations disjoint subsets, each subset of a fragment of the relationship, called partitions; one partition can only exist in a replication group, but a copy groups can host multiple partition between the replication group segmentation operation can move through the levels.



**Vertical partitioning** also known as a collection of vertical partition partition or vertical partitions.

In SequoiaDB cluster environment, the user can also be a collection of properties of the global relations into several subsets, and make the projection operator on these subsets, the subset of these maps to another collection, a collection of relations in order to achieve the sub-vertical shear; the collection is called the main collection, each subset is called segmentation partitions, partition map collection called subsets; one partition can be mapped to a subset of, but a subset can host multiple partitios; partition between the subset of operations can be cut by vertical remapping.



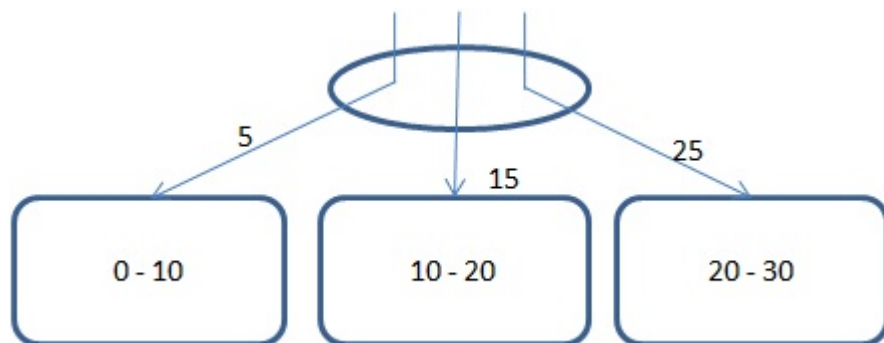
**Mixed partitions** In SequoiaDB clustered environment can be mapped through the first set of vertical partitioning into multiple subsets, and then cut through the horizontal partitioning into multiple subsets replication group in order to achieve a mixed partition.

Partitioning

Whether horizontal or vertical partition partition has two ways: hash partitioning(Hash) and range partitioning(Range). Field is determined in two ways zoning district is based is called "partitioning key". Based on the definition of the partition key collection, each partition key can contain one or more fields.

Under Range partitioning approach based on the scope of the record keys to select the partition you want to insert. Partitioning key records generated hash value selected according to the partition you want to insert Hash under way.

Sharding is based on the definition of collection. Every collection key contains one or more fields.



In this chart, 3 square areas represent 3 data nodes. The ellipse represents coord node. Every data node defines the range of data in it. For example, node 1 stores data between 0 and 10 including 0.

When users insert a record of data, coord node firstly finds out the sharding which the sharding key of the data belongs to. If there is no sharding key, it's defined in the type of Undefined. (Data in Undefined type can also be compared to that in common data types.)

After the sharding is found, coord node will send the request that it receives to it.

### Sharding Key

#### Concept

Sharding key defines the sharding rules of data in every collection. Every collection corresponds a sharding key. A sharding key contains one or more fields.

In catalog node, every collection has its own sharding range. In sharding range, every range segment corresponds to a sharding group, which shows that certain data segment is in the sharding group.



#### Note:

When creating a collection, index key of it is specified. After collection is created, index key is unalterable.

In [sharded collection](#), after a record is inserted into database, the sharding key is not allowed to be updated.

#### Format

- Range sharding Key

The format of sharding key is similar to index key, which is a JSON object. Every field in JSON object corresponds to a field in sharding key. The value is "1" or "-1", respectively representing forward order or reverse order.

```
{ <field 1> : <1|-1> [, <field 2> : <1|-1> ... ] }
```

- Hash Sharding Key

Hash partition Sharding Key composition identical with Range partitioning (but field forward/reverse does not work). Partition represents a hash slice number. The default is  $2^{12}$ , on behalf of our entire range is divided into 4096 the average slice. Design hash fragmentation purpose is to allow data

distribution is more flexible and can be freely set each shard hash slice data partition range. Sharding Type if you do not fill it defaults to Range partition.

```
{ShardingKey: {<field 1>:<1|-1> [, <field 2>:<1|-1>, ...]}, {ShardingType: "hash"} [, {Partition: <number of fragments>}]}
```

### Sample

- A sharding key containing 2 fields, forward sequence and reverse sequence is as follow:

```
{ Field1 : 1, Field2 : -1 }
```

- Hash Sharding Key

```
{{ Field1 : 1, Field2 : -1 }, {ShardingType: "hash"}, {Partition: 2^12}}
```

### Sharded collection

#### Concept

Sharded collection is a collection that has defined a sharding key. Sharded collection can split data in collection to more than one data sharding group according to the fields specified by sharding key.

When collection is generated, users can specify sharding key. Sharded collection is generated in a random data sharding group. User can split a collection to several data sharding groups by themselves.

#### Sharding Interval

Every interval in sharded collection is named sharding interval.

When a sharded collection is generated, the sharding group of it contains all the ranges, which include range from MinKey to MaxKey for all fields.

Every range is left-closed and right-open. That's to say, data in it is greater than or equal to the lower boundary and lesser than the upper boundary. For example:

```
{ LowBound: { "": 10 }, UpBound: { "": 20 } }
```

In this example, the lower boundary is 10 and the high boundary is 20. So all the data in this range for all the sharding field is greater than or equal to 10 and lesser than 20.



**Note:** The definition of all ranges within a collection does not contain field name. The fields in it are consistent with those defined in sharding key in terms of type and number.

When a sharding key contains several fields, the matching principle is to match first field ahead. If it is located in boundary, it will match the next field. For example,

```
{ LowBound: { "": 10, "": 5 }, UpBound: { "": 20, "": 1 } }
```

In this sharding range, if the sharding key users input is located between 10 and 20, it is judged to belong to this range. If it is lesser than 10 or greater than 20, it doesn't belong to this range. If it is 10 or 20, it will compare the second field with the principle of left-closed-right-open.

#### Rule

Please refer to the definition of [SYSCOLLECTIONS](#) for more about the definition principle of sharded collection.

### Sample

A typical sharded range which exists in two sharding group is as follow:

```
[
  { "GroupID" : 1000,
    "LowBound" : { "" : MinKey, "" : MaxKey },
    "UpBound" : { "" : 10, "" : 5 }
  }
```

```

    },
    { "GroupID" : 1001,
      "LowBound" : { "" : 10, "" : 5 },
      "UpBound" : { "" : MaxKey, "" : MinKey }
    }
  ]

```

The replset ID of the first range in this sample is 1000. The sample contains a sharding key with two fields:

- lower boundary : { "" : MinKey, "" : MaxKey }
- upper boundary : { "" : 10, "" : 5 }

The replset ID of the second range in this sample is 1001. The sample contains a sharding key with two fields:

- lower boundary : { "" : 10, "" : 5 }
- upper boundary : { "" : MaxKey, "" : MinKey }

## Sharding Index

### Concept

There is a default index named "\$shard" in every sharded collection. This is sharding index.

Only sharded collections have sharding index.

Sharding index exists in every sharding group in a sharded collection. The order of field definitions and the order of fields are the same as that of sharding key.



#### Note:

Any unique index defined by user should contain all the fields in sharding index. But the order of fields of them may not be the same.

In sharded collection, the field of "\_id" is unique in the sharding, but not globally unique.

### Sample

A typical sharding index is as follow:

```

{
  "IndexDef" : {
    "name" : "$shard",
    "_id" : { "$oid" : "515954bfa88873112fa6bd3a" },
    "key" : { "Field1" : 1, "Field2" : -1 },
    "v" : 0,
    "unique" : false,
    "dropDups" : false,
    "enforced" : false
  },
  "IndexFlag" : "Normal"
}

```

## Background Task

### Concept

Background task is a kind of task that never block front session. It won't stop when a front session pauses.

All the background tasks is recorded in the collection of SYSCAT.SYSTASKS in catalog node. Different types of background tasks may contain different fields.

All the background tasks contain fields as follow:

Field Name	Type	Description
JobType	Integer	Task type, representd: <ul style="list-style-type: none"> <li>0: data split</li> </ul>
Status	Integer	Task status, represents: <ul style="list-style-type: none"> <li>0: prepare</li> <li>1: run</li> <li>2: pause</li> <li>3: cancel</li> <li>4: change metadata</li> <li>9: completed</li> </ul>
CollectionSpace	String	collection space name
Collection	String	collection name

Background task type includes:

- [data split](#)

## Split

### Concept

A sharded collection is created in a random replset. If users want to horizontally expand the collection and allocate it to more than one replset, data should be split.

Split is an approach to transfer online data from one replset to another replset. In the process of data transference, results of queries may not be consistent, but SequoiaDB can guarantee the consistence of data in disks.

Range partitioning and Hash partition contains two kinds of segmentation methods: percentage range segmentation and segmentation. Cut in the range of time\_sharing, Range partition using precise conditions, Hash partition using Partition (number of fragments) conditions. Cut timeshare start condition is a required field, and the end condition is optional condition, end condition defaults to split the source currently contains the maximum data range.

For example:

Hash: `db.foo.bar.split('src', 'dst', {Partition:10}, {Partition:20})`

Range: `db.foo.bar.split('src','dst',{a:10}, {a:20})`

Data segmentation and data on the partition range are closed to the left to follow the principle of the right to open. Namely: {Partition:10}, {Partition:20} Migrating data representative of the range of [10, 20).

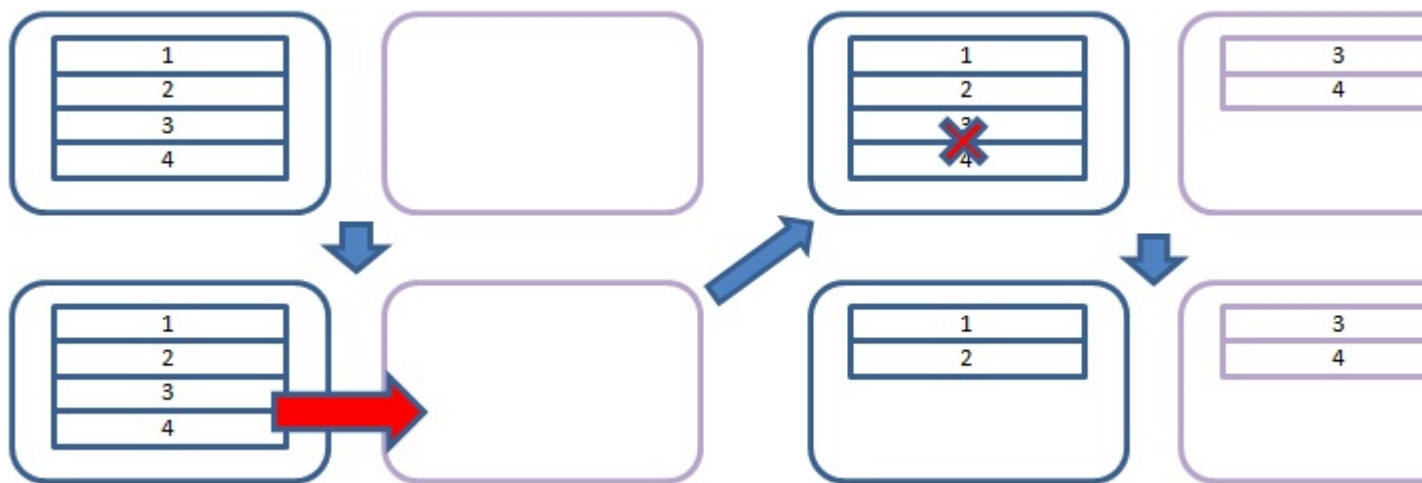
Percentage of segmentation: `db.foo.bar.split('src', 'dst', 50)`



Note:

When the split range does not conflict, segmentation can be done concurrently.

"src"、"dst" represent "the source partition group," "target partition group."



In this chart, the upper left part is the initial status of a system. 4 records are stored in the left node. The split is defined to begin with 3, so data 3 and 4 will be split from the left node, and transferred to the right node (the lower left part of the chart).

The upper right part is the 3rd status, two replsets store the same part of data at the same time. At this moment, data is temporarily inconsistent. But the final status is transformed into the 4th one in the lower right part. Data which is successfully transferred is deleted. Data is consistent finally.

Two replsets communicate with each other in the process of data split:

- source replset: data is originally stored in this replset
- target replset: data is transferred to this replset

### Background Task

Data split is executed by a [background task](#).

This kind of background task contains several special fields:

Field Name	Type	Description
SourceName	String	Source replset name
TargetName	String	Target replset name
SourceID	Integer	Source replset ID
TargetID	Integer	Target replset ID
SplitValue	Object	Data split key

Split is executed through several stages:

#### Prepare Phase

In preparing phase, no task record is inserted into SYSCAT.SYSTASKS of catalog node. The system checks the validity of request in catalog node, and requests the record which meets the specific split condition from source node.

#### Ready Phase

In ready phase, coord node gets sharding key corresponding to split condition from source node, and sends it to catalog node. Catalog node inserts background execution records into the collection called SYSCAT.SYSTASKS.

#### Running Phase

In running phase, catalog node sends split request to target node. Target node



generates a background task, requests data from source node, and sends its status to catalog node. When target node successfully generate a background task, catalog node can immediately continue to work. So catalog node won't be blocked by target node.

#### Clean-up Phase

In clean-up phase, target node has got all data it needs from source node, so it sends clean-up request to catalog node, then source node clean up data that has transferred.

#### Finish Phase

After source node clean up data that has transferred, it informs catalog node. Catalog deletes the task in the collection SYSCAT.SYSTASKS.

## Domain

### Concept

Domain is the logical unit composed of several replica group composed. Each domain can automatically manage data based on defined policies belongs, such as data slice and data isolation.

When the replica group domain is 0 when called airspace. Airspace can not create a collection space.

A replica group can belong to multiple domains.

There is a system known as domain "SYSDOMAIN" logically. The current system all replica groups are part of the system domain. When you create a domain user can not use the "SYSDOMAIN" as a domain name, it can not directly operate the system domain.

Domain has the following attributes in addition to names:

Property name	Description
AutoSplit	When this property is True, hash partitions on the domain to create a collection of all the replica group will be automatic segmentation to contain.

# Install Overview

---

The basic steps to install Sequoiadb are as follow:

[Plan Database Deploy](#)

[System Requirement of SequoiaDB](#)

[Install Media](#)

[SequoiaDB server installation](#)

[System configuration and start](#)

## Overview on Database Deployment

---

SequoiaDB is a completely distributed system architecture. It supports all kinds flexible deployment. In order to improve hardware and software performance, before installing system, we need to plan the deployment and the network connection. SequoiaDB supports 2 approaches of deployment:

- Standalone mode

Start one business process that manipulate data. This mode supports high performance. It is easier to deploy. But the disadvantage is that it doesn't support distributed deployment or high-accessibility. It fits situation that has low amount of data and low total IOPS throughout, and requires low latency in single manipulation.

- Cluster Mode

In this mode, the system can be deployed on multiple physical machines. It supports at most 300 physical machines. In cluster mode, catalog node, data node, coord node and Web management node (optional) should be deployed. As many as 65535 logical nodes can be deployed on one physical machine.



Note: Dependent mode can be shifted to cluster mode. In this process, the system will pause business processes which cost least than 10 minutes.

Users can plan their deployment according to data capacity, performance, reliability, cost. Here are some typical deployment approaches for reference.

Actually, the approach of deployment can be very flexible. Users can combine different kind of deployment to meet their specific needs.

[Easiest Deployment](#)

[High Availability Deployment](#)

[High Performance Deployment](#)

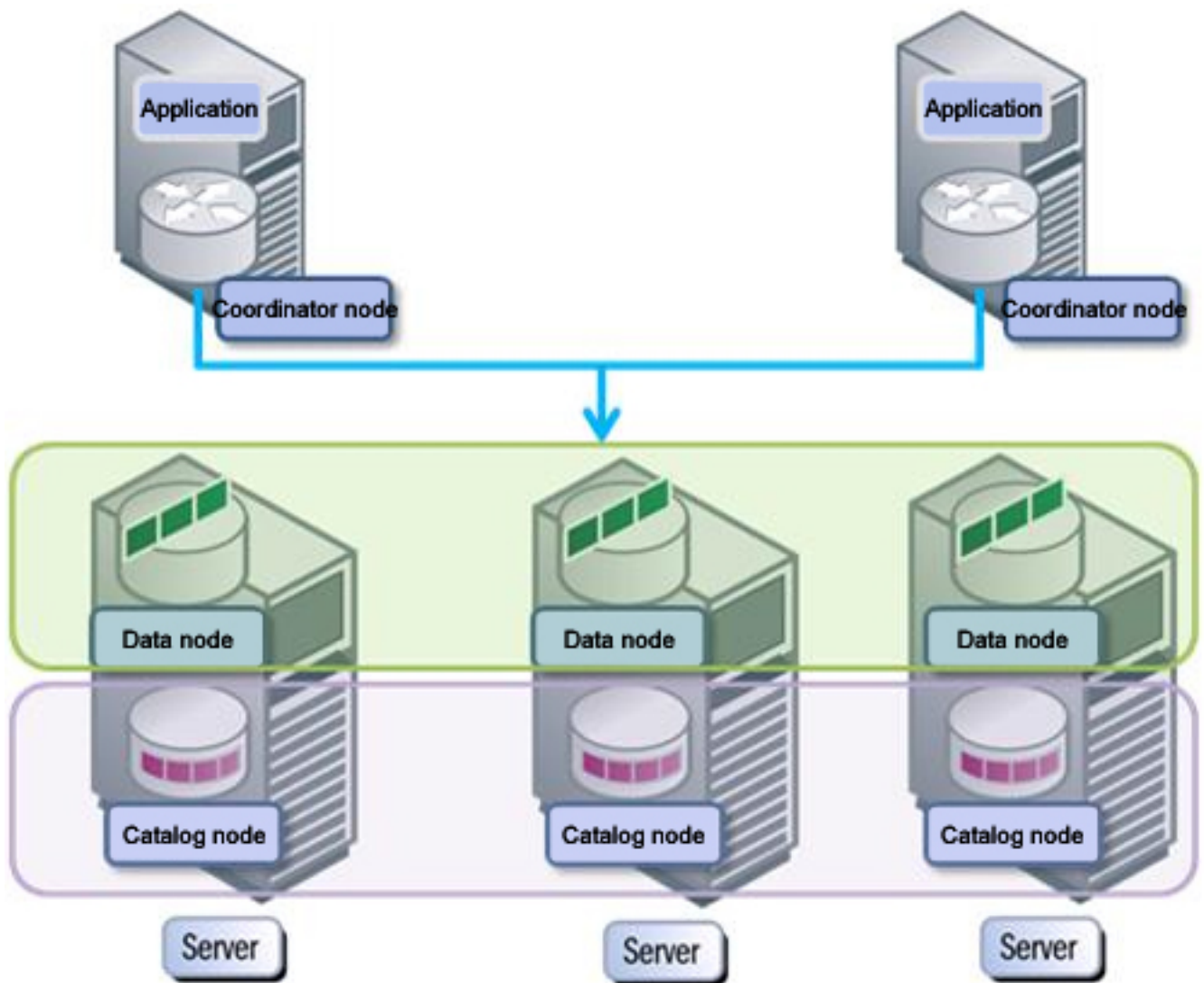
### Easiest deployment

Easiest deploy is proper to undemanding applications that has low requirement to database, small amount of data and low throughput, requiring low reliability. In this mode, SequoiaDB just start one database server process. Application can be put on the same server as database, or deployed in another server.



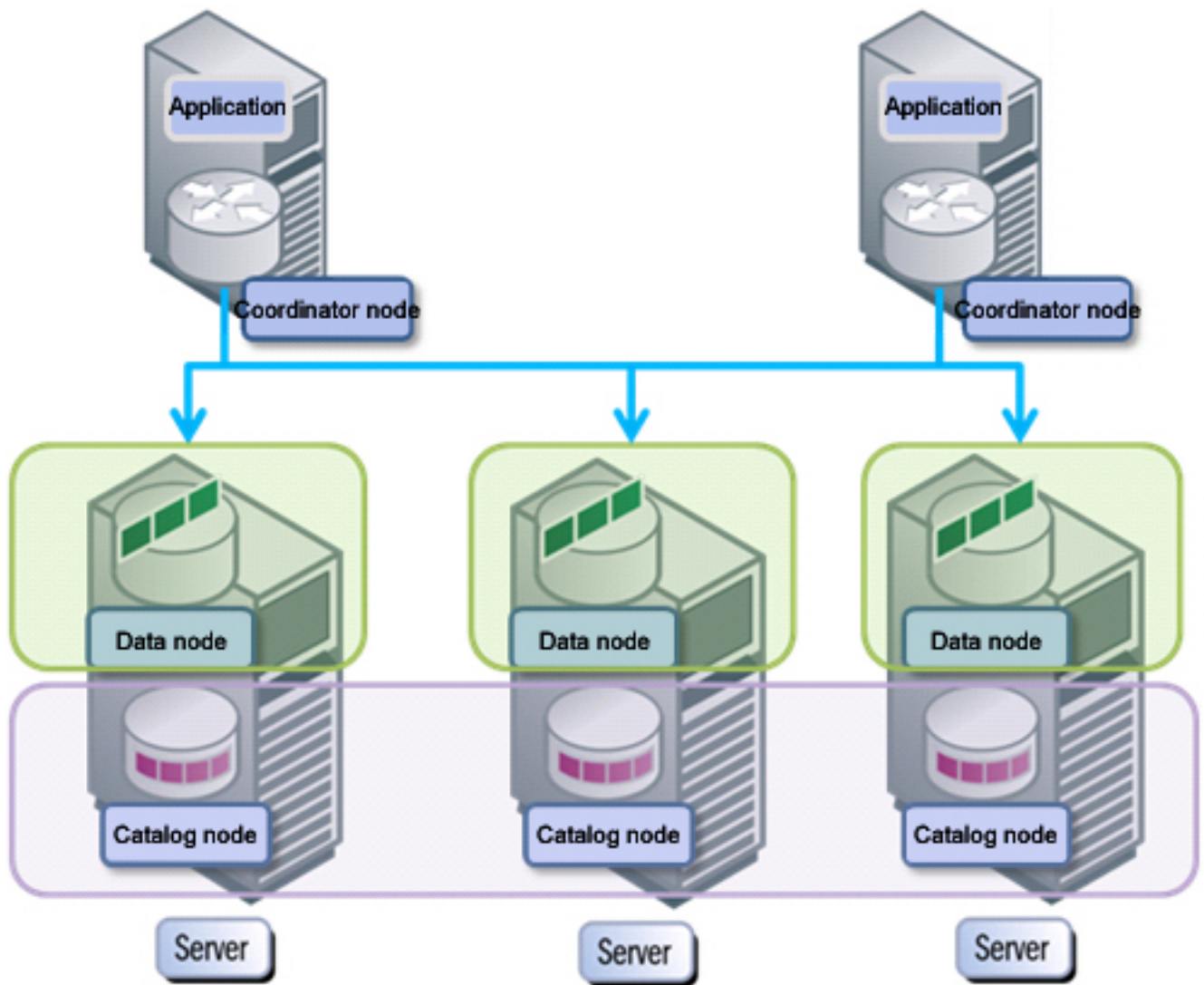
## High-availability Deployment

High-availability deployment fits applications that require high reliability, but low amount of data and low total throughput. In this mode, there are catalog nodes and data nodes in three physical machines. A replset is consisted of 3 data nodes. Every copy cluster is consisted of 3 catalog nodes. Coord nodes and applications should be deployed on different servers or the same database server. The advantage of this mode is high reliability. If any server breaks down, all data will be able to be read or written. But the capacity of data is the same with that of a server, and hardware cost is higher.



### High-performance Deployment

High-performance deployment fits applications that require high total throughput and high performance, low hardware cost but low reliability. In this mode, there are coord nodes and data nodes in three physical machines. A replset is consisted of 3 coord nodes. Every data node makes up one replset (It contains just one copy). Catalog nodes and applications should be deployed on different servers or the same database server. This mode can make full use of all memory in servers. The total memory capacity is the sum of memeory capacities of 3 servers. But the reliability is low. If any server breaks down, some data will fail to be read or written.



## System Requirement of SequoiaDB Installation

Before installing SequoiaDB, please ensure that the system you choose can meet the requirement of OS, hardware, communication, disk and memory. The command "sdbpreck" is used to check the precondition of system to guarantee security.

[Hardware Requirement](#)

[Supported OS](#)

[Software Requirement](#)

### Hardware Requirement

Requirement	Requirement	Suggestion
CPU	Support the following processor <ul style="list-style-type: none"> <li>x86(Intel Pentium、 Intel Xeon and AMD) 32-bit Intel and AMD processor</li> <li>X64(64-bit AMD64 and Intel EM64T processor)</li> <li>PowerPC7 or PowerPC7+ processor</li> </ul>	Suggest to use X64(64-bit AMD64 and Intel EM64T processor ) or PowerPC processor

Requirement	Requirement	Suggestion
Disk	At least 10GB	More than 100GB
Memory	More than 128MB	More than 2GB
Network card	At least 1 network card	At least 1GE network card

## Supported Operating System

Type of system	List of system
Linux	<ul style="list-style-type: none"> <li>Red Hat Enterprise Linux (RHEL) 6</li> <li>SUSE Linux Enterprise Server (SLES) 11 Service Pack 1</li> <li>SUSE Linux Enterprise Server (SLES) 11 Service Pack 2</li> <li>Ubuntu 12</li> <li>CentOS 6</li> </ul>
Power PC Linux	<ul style="list-style-type: none"> <li>Red Hat Enterprise Linux (RHEL) 6</li> <li>SUSE Linux Enterprise Server (SLES) 11 Service Pack 1</li> <li>SUSE Linux Enterprise Server (SLES) 11 Service Pack 2</li> </ul>

## Software Requirement

### Linux System requirement


- System configuration requirement

Configuration Item	Configuration method	Verification method
Set hostname	<ol style="list-style-type: none"> <li>Log in with root privileges, perform hostname sdbserver1(sdbserver1 hostname can be modified as needed);</li> </ol> <ul style="list-style-type: none"> <li>For SUSE:               <ol style="list-style-type: none"> <li>Open the /etc/HOSTNAME file;                   <pre>vi /etc/HOSTNAME</pre> </li> <li>Modify file contents, configured as a hostname;                   <pre>sdbserver1 (hostname)</pre> </li> <li>Press :wq to save and exit;</li> </ol> </li> <li>For RedHat:               <ol style="list-style-type: none"> <li>Open the /etc/sysconfig/network file;                   <pre>vi /etc/sysconfig/network</pre> </li> <li>HOSTNAME line will modify the HOSTNAME=sdbserver1, which sdbserver1 new hostname;</li> <li>Press :wq to save and exit;</li> </ol> </li> <li>For Ubuntu :               <ol style="list-style-type: none"> <li>Open the /etc/hostname file;                   <pre>vi /etc/hostname</pre> </li> <li>Modify file contents, configured as a hostname;                   <pre>sdbserver1</pre> </li> <li>Press :wq to save and exit;</li> </ol> </li> </ul>	Execute hostname command to verify whether the information is printed "sdbserver1"

Configuration Item	Configuration method	Verification method
Enable server nodes to visit each other through hostname	<ul style="list-style-type: none"> <li>Log in the system in the role of root, open "/etc/hosts" with the command "vi /etc/hosts"</li> <li>Alter "/etc/hosts", set the mapping between hostname of server node and IP in this file.  192.168.20.200 sdbserver1  192.168.20.201 sdbserver2  192.168.20.202 sdbserver3</li> <li>3 .Save and exit</li> </ul>	Check with "ping sdbserver1"  Check with "ping sdbserver2"

- Close the firewall

Configuration Item	Configuration method	Verification method
Close the firewall	Close the firewall operation, need administrator privileges. <ul style="list-style-type: none"> <li>For SUSE : 1.SuSEfirewall2 stop ; 2.chkconfig SuSEfirewall2_setup ;</li> <li>For RedHat : 1.service iptables stop ; 2.chkconfig iptables off ;</li> <li>For Ubuntu : 1.ufw disable ;</li> </ul>	<ul style="list-style-type: none"> <li>For SUSE : chkconfig -list   grep fire ;</li> <li>For RedHat : service iptables status ;</li> <li>For Ubuntu : ufw status ;</li> </ul>

 Note: Each database server should be configured.

### Recommended configuration in Linux

- Adjusting ulimit

modify the configuration file "/etc/security/limits.conf ":

```
#<domain>      <type>      <item>      <value>
*               soft       core        0
*               soft       data        unlimited
*               soft       fsize       unlimited
*               soft       rss         unlimited
*               soft       as          unlimited
```

#### Params Description :

core : the "core" file is generated as the time of database failure to troubleshooting.,you had better close the system for advice.

data : The size of data memory that database processes allow to allocate.

fsize : The size of file that database processes allow to address.

rss : The max size of resident set that database processes allowed.

as : The max amount of virtual memory addressing space that database processes allowed.



Note:

1. Each database server should be configured.
  2. It needs to relogin after changing to be effective.
- Adjusting kernel params
    1. Using the following command to output the current configuration of vm, and be placed on file.

```
cat /proc/sys/vm/swappiness
cat /proc/sys/vm/dirty_ratio
cat /proc/sys/vm/dirty_background_ratio
cat /proc/sys/vm/dirty_expire_centisecs
cat /proc/sys/vm/vfs_cache_pressure
cat /proc/sys/vm/min_free_kbytes
```

2. Adding the following params to the file of "/etc/sysctl.conf" to adjust kernel params.

```
vm.swappiness = 0
vm.dirty_ratio = 100
vm.dirty_background_ratio = 10
vm.dirty_expire_centisecs = 50000
vm.vfs_cache_pressure = 200
vm.min_free_kbytes = <8% of physical memory size, unit is KB>
```



Note: When the available physical memory size of database isn't enough 8GB, doesn't need to set "vm.swappiness = 0".

3. Executing the following command to be effective.

```
/sbin/sysctl -p
```



Note: Each database server should be configured.

- Database directory structure

The users had better put the directory of data, index, and log in different physical disks, to reduce the competition between sequential I/O and random I/O.

## Install Media

Please go to official website to download the appropriate version of SequoiaDB.

Download address: <http://www.sequoiadb.com/index.php?p=downserver>

## Installation of SequoiaDB Server

Available system of installation :

Installation	Linux	Windows
Wizard Installation	Yes	Yes

### Wizard Installation

Wizard installation is a GUI installation program available for Linux and Windows. It is easy to install through the GUI program. It is used to show settings and install SequoiaDB server.

In Linux, X Server is essential to show GUI. If there is no X Server in Linux, users can choose to install through text interface.


[Installation of SequoiaDB in Linux](#)



## Installing Linux version on SequoiaDB Server

### Preparation before installation

- Ensure that the system support the requirement of installation, content and disk
- Log in on the role of root to install SequoiaDB database service
- Check the mapping between SequoiaDB package and OS
- If you choose to install through GUI mode, please make sure that X server service is running
- Check whether the server has set hostname, and it can generate network connection with other servers (For example, ssh hostname)


 Note: SequoiaDB installation wizard doesn't support non-English characters.

### Installation steps

#### Explanation

(1) Here we take sequoiadb-1.0.0-linux-x86\_64-installer.run as the sample installation package.

(2) The steps are introduced through command lines. In GUI installation, users can install server program according to image wizard prompts.

 Note: If there are more than one server, users should repeatedly install server program on them one by one according to the following steps.

- Refer [System configuration requirement](#) to configure Host Name and change the system kernel params.
- Run installation program

```
./sequoiadb-1.0.0-linux-x86_64-installer.run --mode text
```

- The system prompt you to choose wizard language

```
Language Selection
Please select the installation language
[1] English - English
[2] Simplified Chinese - 简体中文
Please choose an option [1] :1
```

- Input 1 to choose English, then the system prompt you to input installation catalog

```
-----
Created with an evaluation version of BitRock InstallBuilder
```

```
Welcome to the SequoiaDB Server Setup Wizard.
```

```
-----
Please specify the directory where SequoiaDB Server will be installed.
```

```
Installation Directory [/opt/sequoiadb]:
```

- After inputting the installation path(installed in /opt/sequoiadb in default) .Then you are prompted to enter a user name which used to run sequoiadb service

```
Configure user information
Please insert the desired username and password
User Name [sdbadmin]:
```

- Pressing enter after inputting the user name(creating sdbadmin user in default).Then system prompts you to enter password and confirmed password

```
Password [*****] :
Re-enter [*****] :
```

- After inputting password twice (The default password is sdbadmin), the system will prompt you to set service port

```
-----
SDBCM Port

SequoiaDB Cluster Manager port

Port [11790]:
-----
```



Note: Service ports of all servers should be the same.

- Input port (The default port is 11790). The system prompts the user to confirm the installation began.
- Asked whether to allow sequoiadb processes related to boot from the start.

```
-----
Whether to allow Sequoiadb related processes related to boot from the start.
```

- Sequoiadb processes related to boot from the start [Y/n]:Y Input Y, press enter, sequoiadb related processes to boot from the start.

```
-----
Setup is now ready to begin installing SequoiaDB Server on your computer.
```

```
Do you want to continue? [Y/n]:
```

- Input "Y" and press return key. Then the system begin to install. After about 1 minute, installation is finished

```
Installing SequoiaDB Server to your computer, please wait for a moment.
```

```
Installing
```

```
0% ----- 50% -----100%
#####
```

```
-----
SequoiaDB Server Program has been installed to your computer.
```

## Start SequoiaDB web service

By starting the background Web service of SequoiaDB, can also do operations to database, and convenient for management.

- enter the directory: /opt/sequoiadb/bin
- execute the following command:

```
./sdbwsart -S <server name: port>
```

server name: specify the service IP address, such as "192.168.10.10"

port: specify the service port, such as "8080"

Finally enter "http://192.168.10.10:8080" in the browser, then you can access to the database management page.

## System configuration and startup

Catalog:

[Configuration and startup of standalone mode](#)

[Configuration and startup of cluster mode](#)

## Configuration and startup of standalone mode

Explanation:

(1) This section will take the easiest deployment as an example to introduce configuration and startup steps.

(2) The following manipulation supposes that SequoiaDB is installed under the catalog "/opt/sequoiadb".

(3) All of "sdb" service processes run as "sdbadmin", please ensure all of the database files have the "sdbadmin" read and write permissions.

- switch to "sdbadmin"

```
su sdbadmin
```

- Enter installation catalog (The following content supposes that the default installation path is "/opt/sequoiadb")

```
cd /opt/sequoiadb
```

- Create a catalog to store configuration file

```
mkdir #p /opt/sequoiadb/conf/local/11810
```

(Here 11810 is the port of database service. To avoid port conflicts and other issues, not the database port configuration in the random port range. Such as: Most linux default random port range of 32768~61000 can be configured in the database port 32767 or less.)

- Copy the sample configuraiton of standalone mode

```
cp /opt/sequoiadb/conf/samples/sdb.conf.standalone /opt/sequoiadb/conf/local/11810/sdb.conf
```

- Open the configuration file

```
vi /opt/sequoiadb/conf/local/11810/sdb.conf
```

- Alter nessesary configuration parameters

```
# database path
```

```
dbpath=/opt/sequoiadb/database/standalone
```

The path of database. It is configurated according to requirement, and ensure that the path exists (Please do not manually create it! )

- Type in ":wq" to save and exit.
- Create the path to store data file

```
mkdir #p /opt/sequoiadb/database/standalone
```


This path is the same as that in previous step

- Start database process

```
/opt/sequoiadb/bin/sdbstart #c /opt/sequoiadb/conf/local/11810
```

- Finish configuration and startup of database

## Configuration and startup of cluster mode

 Note: When configuring a cluster mode, make sure that the mapping between the server and the host name is correct, details, refer to the [system configuration needs](#) to ensure that the communication between the nodes, the nodes of the firewall is turned off.

Instruction :

(1) This section will introduce the steps of configuration and startup by taking highly accessible deployment as an example.

(2) The following steps are based on the fact that SequoiaDB is installed under the catalog "/opt/sequoiadb".

(3) All of "sdb" service processes run as "sdbadmin", please ensure all of the database files have the "sdbadmin" read and write permission.

- Step 1: Check the configuration service status of SequoiaDB.

Check the configuration service status of SequoiaDB in each database server: `service sdbcm status`

If the system prompts "sdbcm is running", we can know that service is running, or please run the following command to reconfigure service program: `service sdbcm start`

- Step 2: start one coord node

1. switch to "sdbadmin"

```
su sdbadmin
```

2. On any database server (the following steps are manipulated on this server), create the coord node configuration catalog

```
mkdir -p /opt/sequoiadb/conf/local/11810
```

Here "11810" is the service port of coord node. It can be set according to requirement.

3. Copy sample configuration file of coord node

```
cp /opt/sequoiadb/conf/samples/sdb.conf.coord /opt/sequoiadb/conf/local/11810/sdb.conf
```

4. Alter configuration file

```
vi /opt/sequoiadb/conf/local/11810/sdb.conf
```

Alter content

```
# database path
```

```
dbpath=/opt/sequoiadb/database/coord
```

This is the path of database. It can be changed according to requirement. Please ensure that the path exists (Please create manually if it doesn't exist.)

5. Type in ":wq" to save and exit

6. Create path that stores data file

```
mkdir -p /opt/sequoiadb/database/coord
```

The path is the path in previous step

7. Start coord node process

```
/opt/sequoiadb/bin/sdbstart -c /opt/sequoiadb/conf/local/11810/
```

- Step 3: Configure and start catalog nodes with statements

1. Start SequoiaDB Shell console on the physical machine of coord node

```
/opt/sequoiadb/bin/sdb
```

2. Connect to coord node

Input in shell:

```
> var db = new Sdb("localhost", 11810)
```

Here "11810" is the service port of coord node.

3. Create a catalog node group

```
> db.createCataRG("sdbserver1", 11800, "/opt/sequoiadb/database/cata/11800")
```

Here "sdbserver1" is the hostname of the 1st server.

Here "11800" is the service port of catalog node. (The port is configured not to conflict with a random port, other ports in the following configuration also requires attention.

" /opt/sequoiadb/database/cata/11800" is the path of data file of catalog node.

#### 4.Wait for 5 second, and add 2 more catalog nodes

```
>var cataRG = db.getRG("SYSCatalogGroup");
>var node1 = cataRG.createNode("sdbserver2", 11800, "/opt/sequoiadb/database/cata/11800")
>var node2 = cataRG.createNode("sdbserver3", 11800, "/opt/sequoiadb/database/cata/11800")
```

#### 5.Start catalog node group

```
>node1.start()
>node2.start()
```



#### Note:

When creating a node, the first parameter should be "hostname", rather than IP of the host.

- Step 4: Configure and start data nodes with statements

#### 1.Create data node group

```
>var dataRG = db.createRG("datagroup")
```

#### 2.Add data node

```
>dataRG.createNode("sdbserver1", 11820, "/opt/sequoiadb/database/data/11820")
>dataRG.createNode("sdbserver2", 11820, "/opt/sequoiadb/database/data/11820")
>dataRG.createNode("sdbserver3", 11820, "/opt/sequoiadb/database/data/11820")
```



#### Note:

When creating a node, the first parameter should be "hostname", rather than IP of the host.

#### 3.Start data node group

```
>dataRG.start()
```

#### 4.Exit SequoiaDB shell console

```
>quit
```

- Step 5: Start coord node on other 2 servers

#### 1.Create configuration catalog of coord node

```
mkdir -p /opt/sequoiadb/conf/local/11810
```

Here "11810" is the service port of coord node. It can be set according to requirement.

#### 2.Copy sample configuration file of coord node

```
cp /opt/sequoiadb/conf/samples/sdb.conf.coord /opt/sequoiadb/conf/local/11810/sdb.conf
```

#### 3.Alter configuration file

```
vi /opt/sequoiadb/conf/local/11810/sdb.conf
Alter content
# database path
dbpath=/opt/sequoiadb/database/coord This is the path of database. It can be changed
according to requirement.
Please ensure that the path exists. (Please create manually if it doesn't exist.)

the following line
# catalog addr (hostname1: servicename1, hostname2: servicename2,...)
#catalogaddr=
modify to
# catalog addr (hostname1: servicename1, hostname2: servicename2,...)
```

```
catalogaddr=sdbserver1:11803,sdbserver2:11803,sdbserver3:11803 This id the address and port of Catalog service.
```

4.Type in ":wq" to save and exit

5.Create path of data file

```
mkdir -p /opt/sequoiadb/database/coord This path is the one in previous step.
```

6.Start coord node process

```
/opt/sequoiadb/bin/sdbstart -c /opt/sequoiadb/conf/local/11810/
```

End

## The Test Environment

---

1.Enter the SequoiaDB shell console.

```
/opt/sequoiadb/bin/sdb
```

2.Create a new sdb connection.

```
db = new Sdb("localhost",11810);
```

3.Create a collection of space.

```
db.createCS("foo");
```

4.Create a collection.

```
db.foo.createCL("bar");
```

5.Written records.

```
db.foo.bar.insert({"name": "sequoiadb"});
```

6.Query results.

```
db.foo.bar.find();
{
  "_id": {
    "$oid": "53a82aa2c4b970091e000000"
  },
  "name": "sequoiadb"
}
Return 1 row(s).
```

The query results are correct.

## Uninstall

---

- Log in as root database server
- Run the following command to stop SequoiaDB Configuration Service program

```
service sdbcm stop
```

- Run the following command to stop the database service program SequoiaDB

```
/opt/sequoiadb/bin/sdbstop
```

- Run the following command to uninstall the software SequoiaDB

```
/opt/sequoiadb/uninstall
```

- Fallback system configuration parameters

1.delete the params in the file "/etc/security/limits.conf "

```
# <#domain>      <type>      <item>      <value>
# *                soft        core        0
```

```
# *          soft    data    unlimited
# *          soft    fsize   unlimited
# *          soft    rss      unlimited
*           soft    as       unlimited
```

2.delete the params in the file "/etc/sysctl.conf"

```
vm.swappiness = 0
vm.dirty_ratio = 100
vm.dirty_background_ratio = 10
vm.dirty_expire_centisecs = 50000
vm.vfs_cache_pressure = 200
vm.min-free-kbytes = <8% of physical memory size,unit is KB>
```

# DataBase Administration

database administration related content.

## DataBase Management

This section provides information about basic management tasks.

### Database Runtime Configurion

#### Parameter Instruction

Parameter Name	Acronym	Type	Description
--help	-h	--	Print help information.
--dbpath	-d	str	1. It specifies the path of data file. 2. If it is not specified, the default path is current path.
--indexpath	-i	str	1. It specifies the path of index file. 2. If it is not specified, the default path is the same with 'dbpath'.
--confpath	-c	str	1. Specified configuration file path (doesn't contain fiel name). The system will find under confpath. 2. contains necessary configuration items. The format of a configuration item is "parameter name=value" like "svcname=11810" and "diaglevel=3". 3. If this parameter is not specified, system will automatically search for sdb.conf in current path. 4. There may not be sdb.conf.
--logpath	-l	str	1. Synchronous log file willbe generated when replication node does full sync. This parameter is used to specified the path of synchronous log file. 2. If it is not specified, the default path is "Data file path/replicalog".
--diagpath	--	str	1. It specifies the catalog of diagnostic log. 2. If it is not specified, the default path is "Data file path/diaglog".
--diagnum	--	num	1. It specifies the number of diagnostic log file. 2. If it is not specified, the default 20; -1 means unlimited.
--bkuppath	--	str	1. It specifies the catalog of backup file. 2. If it is not specified, the default path is "Data file path/bakfile".



Parameter Name	Acronym	Type	Description
--maxpool	--	str	1. It specifies the amount of thread in thread pool. 2. If it is not specified, the default value is 0.
--svcname	-p	str	1. It specifies local service port. 2. If it is not specified, the default value are 11800 for coord, 11810 for catalog, 11820 for datanode.
--replname	-r	str	1. It specifies data synch port. 2. If it is not specified, the default value is "svcname+1".
--shardname	-a	str	1. It specifies shard port. 2. If it is not specified, the default value is "svcname+2".
--catalogname	-x	str	1. It specifies catalog port. 2. If it is not specified, the default value is "svcname+3".
--httpname	-s	str	1. It specifies http port. 2. If it is not specified, the default value is "svcname+4".
--diaglevel	-v	num	1. It specifies the print level of diagnostic log. Diagnostic log in sequoiaDB, 0-5 respectively represent: SEVERE, ERROR, EVENT, WARNING, INFO, DEBUG. 2. If it is not specified, the default value is "WARNING".
--role	-o	str	1. It specifies serving role. In sequoiaDB, "data/coord/catalog/standalone" represents "data node/coord node/catalog node/standalone machine". 2. If it is not specified, the default value is "standalone".
--catalogaddr	-t	str	1. It specifies the address of catalog node. The format is "hostname1:catalogname1,hostname2:catalogname2,...". 2. It should specify address of at least one catalog.
--logfilesz	-f	num	1. It specifies the size of sync log file. It is between 64(MB) and 2048(MB). 2. If it is not specified, the default value is 64 (MB).
--logfilenum	-n	num	1. It specifies the amount of synchronous log file. 2. If it is not specified, the default value is 20.
--transactionon	-e	boolean	1. It specifies whether transaction turns on or not. 2. If it is not specified, the default value is false.

Parameter Name	Acronym	Type	Description
--numpreload	--	num	number of page pre-loaders, default:0, value range:[0,100]
--maxprefpool	--	num	maximum number of prefetchers, default:200, value range:[0,1000]
--maxsubquery	--	num	maximum number of query's sub-query, default:10, min value is 0, max value can't more than param 'maxprefpool'.
--maxreplsync	--	num	maximum number of repl-sync, default:10, value range:[0, 200], 0:disable concurrent repl-sync.
--logbuffsize	--	num	the replica-log's memory page number,default is 1024, value range:[512,1024000], but total log memory size can't more than all log file size; a page size is 64KB.
--tmppath	--	num	The temp path of db, default is 'dbpath'+'/tmp'.
--sortbuf	--	num	size of the sorting buf(MB), default:512, min value:128.
--hjbuf	--	num	size of the hash join buf(MB), default:128, min value:64.
--syncstrategy	--	str	The control strategy of data sync in repl group, value enumeration: none,keepnormal,keepall, default:keepnormal.
--preferedinstance	--	str	1. It specifies the instance on which to query. 2. If it is not specified, the default value is anyone. 3. value enum: M--read and write instance, S--read only instance, A--anyone instance, 1-7--the nth instance.
--numpagecleaners	--	num	The number of page cleaners to start during database startup.0 means do not start any page cleaners, default 1.
--pagecleaninterval	--	num	The minimum interval between two cleans for each CS. Unit: ms, Default: 10000, Min: 1000.

**Note:**

SequoiaDB supports setting configuration with command line,configuration file, or both of them. When both of them are used, parameters in command line will overwrite those in configuration file.

The total size of synch log (logfilesz \* logfilenum)determines the fault tolerance in the process of sync. If log is bigger, the possibility of full sync is lower.

## Monitoring

### Concept

Monitoring is an approach to monitor the status of current system. In SequoiaDB, users can monitor system with SNAPSHOT command and LIST command.



Note: If the query is a snapshot in a clustered environment, you can get connected to the coordinator node.

Coordinator node connections, the default is to get a snapshot of the entire cluster of information, for example: snapshot(SDB\_SNAP\_SYSTEM)

To obtain the specified partition group snapshot information, using criteria query, for example: snapshot(SDB\_SNAP\_SYSTEM,{ GroupName: "group1" } )

To get a snapshot of the information specified node, using criteria query, for example: snapshot(SDB\_SNAP\_SYSTEM,{ HostName: "host1", svcname: "11820" } )

### Snapshot

Snapshot is a kind of command which can get the current status of system. There are different types of snapshots as follow:

Snapshot Sign	Snapshot Type	Description
<a href="#">SDB_SNAP_CONTEXTS</a>	Context	Context snapshot contains all the contexts corresponding to all the sessions in current data node.
<a href="#">SDB_SNAP_CONTEXTS_CURRENT</a>	Context of current session	Current contexts snapshot contains the session contexts corresponding to current connections in data nodes.
<a href="#">SDB_SNAP_SESSIONS</a>	Session	Session snapshot contains all the sessions between users and system on current database node.
<a href="#">SDB_SNAP_SESSIONS_CURRENT</a>	Current session	Current session snapshot contains the current sessions on current database node.
<a href="#">SDB_SNAP_COLLECTIONS</a>	Collection	Collections snapshot contains all the non-temporary collections in current data nodes or current cluster.
<a href="#">SDB_SNAP_COLLECTIONSPACES</a>	Collection space	Collection space snapshot contains all the collection space in the current data node or cluster. (except from catalog collection space)
<a href="#">SDB_SNAP_DATABASE</a>	Database	Database snapshot contains database monitoring information of current database node.
<a href="#">SDB_SNAP_SYSTEM</a>	System	System snapshot contains system monitoring information of current database node.
<a href="#">SDB_SNAP_CATALOG</a>	Catalog information	View catalog information for.

### List

List is a kind of lightweight command for getting system current status. There are different types of lists as follow:

List Sign	List Type	Description
<a href="#">SDB_LIST_CONTEXTS</a>	Context	Context list contains all the contexts corresponding to all the sessions in current data node.
<a href="#">SDB_LIST_CONTEXTS_CURRENT</a>	Context of current session	Current contexts list contains the session contexts corresponding to current connections in data nodes.
<a href="#">SDB_LIST_SESSIONS</a>	Session	Session list contains all the sessions between users and system on current database node.
<a href="#">SDB_LIST_SESSIONS_CURRENT</a>	Current session	Current session list contains the current sessions on current database node.
<a href="#">SDB_LIST_COLLECTIONS</a>	Collection	Collections list contains all the non-temporary collections in current data nodes or current cluster.
<a href="#">SDB_LIST_COLLECTIONSPACES</a>	Collection space	Collection space snapshot contains all the collection space in the current data node or current cluster. (expect from catalog collection space)
<a href="#">SDB_LIST_STORAGEUNITS</a>	Storage unit	Storage unit list contains information of all storage units on current database node.
<a href="#">SDB_LIST_GROUPS</a>	Replica Group	Replica Group list contains information of all shards in current cluster.

## Snapshot

This section introduces information of all kinds of snapshots:

[Contexts Snapshot](#)

[Current Contexts Snapshot](#)

[Session Snapshot](#)

[Current Session Snapshot](#)

[Collections Snapshot](#)

[Collection Spaces Snapshot](#)

[Database Snapshot](#)

[System Snapshot](#)

[Catalog Snapshot](#)

Context Snapshot

## Description

Context snapshot contains all the contexts corresponding to all the sessions in current data node.

A session is a record. If a session contains one or more contexts, the Contextd array field generate a for each of the contexts.



Note: The manipulation of snapshot will generate a context, so the result set will contains at least one context of current snapshot.

## Sign

[SDB\\_SNAP\\_CONTEXTS](#)

## Proper Node

data node,catalog node

### Field Information

Field Name	Type	Description
SessionID	String	Session ID(HostName:Port:ID)
Contexts.ContextID	Long integer	Context ID
Contexts.DataRead	Long integer	Data that is read
Contexts.IndexRead	Long integer	Index that is read
Contexts.QueryTimeSpent	Float	Total time of query (second)
Contexts.StartTimestamp	Timeline	Start time

## Sample

```
> db.snapshot (SDB_SNAP_CONTEXTS)
{
  "SessionID": "vmsvr2-suse-x64:11820:28",
  "Contexts": [
    {
      "ContextID": 12,
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-09-27-18.06.37.079570"
    }
  ]
}
```

## Current Contexts Snapshot

### Description

Current contexts snapshot contains the session contexts corresponding to current connections in data nodes.

It returns one record. In a record, Contexts array field contains all the contexts in current session.



**Note:** The manipulation of snapshot will generate a context, so the result set of it at least contains one context.

## Sign

SDB\_SNAP\_CONTEXTS\_CURRENT

## Proper Node

data node,catalog node

### Field Information

Field Name	Type	Description
SessionID	String	SessionID(HostName:Port:ID)
Contexts.ContextID	Long integer	Context ID
Contexts.DataRead	Long integer	Data that is read
Contexts.IndexRead	Long integer	Index that is read

Field Name	Type	Description
Contexts.QueryTimeSpent	Float	Total query time (second)
Contexts.StartTimestamp	Timeline	Creating time

### Sample

```
> db.snapshot (SDB_SNAP_CONTEXTS_CURRENT)
{
  "SessionID": vmsvr2-suse-x64:11820:28,
  "Contexts": [
    {
      "ContextID": 13,
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-09-27-18.25.17.311168"
    }
  ]
}
```

### Session Snapshot

#### Description

Session snapshot contains all the sessions between users and system on current database node. Each session is recorded as one record.

#### Sign

SDB\_SNAP\_SESSIONS

#### Proper Node

data node,catalog node

#### Field Information

Field Information	Type	Description
SessionID	Integer or long integer	Session ID(Hostname:Port:ID)
TID	Integer	System thread ID corresponding to the session
Status	String	Session status: <ul style="list-style-type: none"> <li>• Creating : creating status</li> <li>• Running : running status</li> <li>• Waiting : waiting status</li> <li>• Idle : idle status of thread pool</li> <li>• Destroying : destroying status</li> </ul>
Type	String	<a href="#">EDU Type</a>
Name	String	EDU name. Generally, it is null
QueueSize	Integer	The length of queue of requests waiting to be processed
ProcessEventCount	Long integer	The amount of requests which have been processed
Contexts	Long integer or array	Context ID array. It is a list containing all contexts in the session
TotalDataRead	Long integer	Total data read

Field Information	Type	Description
TotalIndexRead	Long integer	Total index read
TotalDataWrite	Long integer	Tota data record write
TotalIndexWrite	Long integer	Total index write
TotalUpdate	Long integer	Total amount of updated records
TotalDelete	Long integer	Total amount of deleted records
TotalInsert	Long integer	Total amount of inserted records
TotalSelect	Long integer	Total amount of selected records
TotalRead	Long integer	Total data read
TotalReadTime	Long integer	Total data read time (millisecond)
TotalWriteTime	Long integer	Total data write time (millisecond)
ConnectTimestamp	Timeline	The time of beginning connection
UserCPU	Float	User CPU (second)
SysCPU	Float	System CPU (second)

### Sample

```
> db.snapshot (SDB_SNAP_SESSIONS)
{
  "SessionID": "vmsvr2-suse-x64:11820:1",
  "TID": 8680,
  "Status": "Running",
  "Type": "LogWriter",
  "Name": "",
  "QueueSize": 0,
  "ProcessEventCount": 1,
  "Contexts": [],
  "TotalDataRead": 0,
  "TotalIndexRead": 0,
  "TotalDataWrite": 0,
  "TotalIndexWrite": 0,
  "TotalUpdate": 0,
  "TotalDelete": 0,
  "TotalInsert": 0,
  "TotalSelect": 0,
  "TotalRead": 0,
  "TotalReadTime": 0,
  "TotalWriteTime": 0,
  "ConnectTimestamp": "2013-09-27-13.28.38.927465",
  "UserCPU": "0.410000",
  "SysCPU": "0.150000"
}
{
  "SessionID": "vmsvr2-suse-x64:11820:2",
  "TID": 8681,
  "Status": "Waiting",
  "Type": "DpsRollbackTask",
  "Name": "",
  "QueueSize": 0,
  "ProcessEventCount": 1,
  "Contexts": [],
  "TotalDataRead": 0,
  "TotalIndexRead": 0,
  "TotalDataWrite": 0,
  "TotalIndexWrite": 0,
  "TotalUpdate": 0,
  "TotalDelete": 0,
```

```

    "TotalInsert": 0,
    "TotalSelect": 0,
    "TotalRead": 0,
    "TotalReadTime": 0,
    "TotalWriteTime": 0,
    "ConnectTimestamp": "2013-09-27-13. 28. 38. 927406",
    "UserCPU": "0. 300000",
    "SysCPU": "0. 130000"
  }
  {
    "SessionID": "vmsvr2-suse-x64: 11820: 3",
    "TID": 8682,
    "Status": "Waiting",
    "Type": "Cluster",
    "Name": "",
    "QueueSize": 0,
    "ProcessEventCount": 71977,
    "Contexts": [],
    "TotalDataRead": 0,
    "TotalIndexRead": 0,
    "TotalDataWrite": 0,
    "TotalIndexWrite": 0,
    "TotalUpdate": 0,
    "TotalDelete": 0,
    "TotalInsert": 0,
    "TotalSelect": 0,
    "TotalRead": 0,
    "TotalReadTime": 0,
    "TotalWriteTime": 0,
    "ConnectTimestamp": "2013-09-27-13. 28. 39. 127611",
    "UserCPU": "2. 050000",
    "SysCPU": "1. 010000"
  }...
  {
    "SessionID": "vmsvr2-suse-x64: 11820: 28",
    "TID": 9430,
    "Status": "Running",
    "Type": "Agent",
    "Name": "127. 0. 0. 1: 60309",
    "QueueSize": 0,
    "ProcessEventCount": 9,
    "Contexts": [
      14
    ],
    "TotalDataRead": 0,
    "TotalIndexRead": 0,
    "TotalDataWrite": 0,
    "TotalIndexWrite": 0,
    "TotalUpdate": 0,
    "TotalDelete": 0,
    "TotalInsert": 0,
    "TotalSelect": 0,
    "TotalRead": 0,
    "TotalReadTime": 0,
    "TotalWriteTime": 0,
    "ConnectTimestamp": "2013-09-27-18. 06. 25. 961090",
    "UserCPU": "0. 030000",
    "SysCPU": "0. 060000"
  }

```



## Current Session Snapshot

## Description

Current session snapshot contains the current sessions on current database node. Each session is recorded as one record.

## Sign

SDB\_SNAP\_SESSIONS\_CURRENT

## Proper Node

data node,catalog node

## Field Information

Field Name	Type	Description
SessionID	Integer or long integer	Session ID(Hostname:Port:ID)
TID	Integer	System thread ID corresponding to the session
Status	String	Session status: <ul style="list-style-type: none"> <li>• Creating : creating status</li> <li>• Running : running status</li> <li>• Waiting : waiting status</li> <li>• Idle : idle status of thread pool</li> <li>• Destroying : destroying status</li> </ul>
Type	String	<a href="#">EDU Type</a>
Name	String	EDU name. Generally, it is null
QueueSize	Integer	The length of queue of requests waiting to be processed
ProcessEventCount	Long integer	The amount of requests which have been processed
Contexts	Long integer or array	Context ID array. It is a list containing all contexts in the session
TotalDataRead	Long integer	Total data read
TotalIndexRead	Long integer	Total index read
TotalDataWrite	Long integer	Total data record write
TotalIndexWrite	Long integer	Total index write
TotalUpdate	Long integer	Total amount of updated records
TotalDelete	Long integer	Total amount of deleted records
TotalInsert	Long integer	Total amount of inserted records
TotalSelect	Long integer	Total amount of selected records
TotalRead	Long integer	Total data read
TotalReadTime	Long integer	Total data read time (millisecond)
TotalWriteTime	Long integer	Total data write time (millisecond)
ConnectTimestamp	Timeline	The time of beginning connection
UserCPU	Float	User CPU (second)
SysCPU	Float	System CPU (second)

Sample

```
> db.snapshot (SDB_SNAP_SESSIONS_CURRENT)
{
  "SessionID": "vmsvr2-suse-x64:11820:28",
  "TID": 9430,
  "Status": "Running",
  "Type": "Agent",
  "Name": "127.0.0.1:60309",
  "QueueSize": 0,
  "ProcessEventCount": 12,
  "Contexts": [
    15
  ],
  "TotalDataRead": 0,
  "TotalIndexRead": 0,
  "TotalDataWrite": 0,
  "TotalIndexWrite": 0,
  "TotalUpdate": 0,
  "TotalDelete": 0,
  "TotalInsert": 0,
  "TotalSelect": 0,
  "TotalRead": 0,
  "TotalReadTime": 0,
  "TotalWriteTime": 0,
  "ConnectTimestamp": "2013-09-27-18.06.25.961090",
  "UserCPU": "0.910000",
  "SysCPU": "2.060000"
}
```

Collection Snapshot

Description

Collections snapshot contains all the non-temporary collections in current data nodes (It contains user collection in coord node). Each collection is recorded as a record.

Sign

SDB\_SNAP\_COLLECTIONS

Proper Node

All nodes

Field Information

Since information stored in data nodes and catalog node is not the same, collections list will return different structures according to the kind of the node:

Other node field information

Field Name	Type	Description
Name	String	Full name of collection
Details.ID	Integer	Collection ID,Range from 0 to 4095,Unique in collection space
Details.LogicalID	Integer	Collection logical ID
Details.Sequence	Integer	Sequence number
Details.Indexes	Integer	The amount of indexes in the collection
Details.Status	String	The current status of the collection

Field Name	Type	Description
		<ul style="list-style-type: none"> <li>• Free</li> <li>• Normal</li> <li>• Dropped</li> <li>• Offline Reorg Shadow Copy Phase</li> <li>• Offline Reorg Truncate Phase</li> <li>• Offline Reorg Copy Back Phase</li> <li>• Offline Reorg Rebuild Phase</li> </ul>
NodeName	String	NodeName(HostName+Port)
GroupName	String	The group name where the node in(Only display in the cluster mode)

### Coord Node Field Information

Field Name	Type	Description
Name	String	Full name of collection
Details.GroupName	String	Group name where the node in
Details.Group.ID	Integer	Collection ID,Range 0~4096,Unique in the collectionspace
Details.Group.LogicalID	Integer	Collection logical ID
Details.Group.Sequence	Integer	Sequence number
Details.Group.Indexes	Integer	The number of indexes in the collection
Details.Group.Status	String	The current status of the collection <ul style="list-style-type: none"> <li>• Free</li> <li>• Normal</li> <li>• Dropped</li> <li>• Offline Reorg Shadow Copy Phase</li> <li>• Offline Reorg Truncate Phase</li> <li>• Offline Reorg Copy Back Phase</li> <li>• Offline Reorg Rebuild Phase</li> </ul>
Details.Group.NodeName	String	Node name(hostname+port)

### Sample of other nodes:

```
> db.snapshot (SDB_SNAP_COLLECTIONS)
{
  "Name": "foo. bar",
  "Details": [
    {
      "ID": 0,
      "LogicalID": 0,
      "Sequence": 1,
      "Indexes": 8,
      "Status": "Normal",
      "NodeName": "vmsvr2-suse-x64:11820",
      "GroupName": "datagroup1"
    }
  ]
}
```

**Sample of coord node:**

```
> coord.snapshot (SDB_SNAP_COLLECTIONS)
{
  "Name": "susefoo.susebar",
  "Details": [
    {
      "GroupName": "datagroup1",
      "Group": [
        {
          "ID": 0,
          "LogicalID": 0,
          "Sequence": 1,
          "Indexes": 1,
          "Status": "Normal",
          "NodeName": "vmsvr2-suse-x64:11820"
        }
      ]
    }
  ]
}
```

**Collection Space Snapshot****Description**

Collection space snapshot contains all the collection space in the current data node. Each collection space is recorded as one record.

**Sign**

SDB\_SNAP\_COLLECTIONSPACES

**Proper Node**

All nodes

**Field Information**

Since information stored in data nodes and catalog node is not the same, collection space snap will return different structures according to the kind of the node:

**Other node field information**

Field Name	Type	Description
Name	String	Collection space name
Collection	String array	All the collections in collection space
PageSize	Integer	Size of data page in collection space
GroupName	String	group name where collectionspace in

**Coord Node Field Information**

Field Name	Type	Description
Name	String	Collection space name
Collection	String array	All the collection in the collection space
PageSize	Integer	Size of data page in collection space
Group.GroupID	Integer array	ID list of the replset of the collection space
Group.GroupName	String	Group name list of the replset of the collection space

### Other nodes sample

```
> db.snapshot (SDB_SNAP_COLLECTIONSPACES)
{
  "Collection": [
    {
      "Name": "bar"
    }
  ],
  "PageSize": 4096,
  "Name": "foo",
  "GroupName": "datagroup1"
}
```

### Coord node sample

```
> coord.snapshot (SDB_SNAP_COLLECTIONSPACES)
{
  "Collection": [
    {
      "Name": "bar"
    }
  ],
  "Group": [
    {
      "GroupID": 1000,
      "GroupName": "datagroup1"
    }
  ],
  "Name": "foo",
  "PageSize": 4096,
  "_id": {
    "$oid": "52451a6bf80be5d22b27489b"
  }
}
```

## Database Snapshot

### Description

Database snapshot contains main status and performance monitoring parameter in current database node. It returns one record.

### Sign

SDB\_SNAP\_DATABASE

### Proper nodes

All Node

### Field Information

Field Name	Type	Description
HostName	String	The host name of physical node of database node.
ServiceName	String	Service name specified by svcname. HostName and service name are the sign of a logical node.
NodeName	String	Node name, in the format of <HostName>:<ServiceName>

Field Name	Type	Description
GroupName	String	The name of replset of the logical node. There is no this field under independent mode.
IsPrimary	Bool	IsPrimary shows whether the node is a main node. There is no this field under independent mode.
ServiceStatus	Bool	ServiceStatus represents the status of whether it is available to serve. For example, <a href="#">Full Sync</a> will make it false.
CurrentLSN.Offset	Long integer	Excursion of current LSN
CurrentLSN.Version	Integer	Verssion number of the current LSN
Version.Major	Integer	Major version number of database
Version.Minor	Integer	Minor version number of database
Version.Release	Integer	Released version number of database
CurrentActiveSessions	Integer	Current active session. It contains User EDU and system EDU.
CurrentIdleSessions	Integer	Current inactive session. Generally, inactive session means that EDU is stored in thread pool and waiting to run.
CurrentSystemSessions	Integer	Current system session. CurrentActiveSessions is the amount of current user EDUs.
CurrentContexts	Integer	Amount of current contexts
ReceivedEvents	Integer	Total amount of event requests received by current shard
Role	String	The role of current node
Disk.DatabasePath	String	The path of database
Disk.LoadPercent	Integer	The percentage of memory in databse path in disk
Disk.TotalSpace	Long integer	Total space (byte) in the database path
Disk.FreeSpace	Long integer	Free space (byte) in the database path Ipimportant : this field and above all fields only display in data node and catalog node
TotalNumConnects	Integer	Total amount of database connection requests
TotalDataRead	Long integer	Total data read requests
TotalIndexRead	Long integer	Total index read requests
TotalDataWrite	Long integer	Total data write requests
TotalIndexWrite	Long integer	Total index write requests
TotalUpdate	Long integer	Total amount of updated records
TotalDelete	Long integer	Total amount of deleted records
TotalInsert	Long integer	Total amount of inserted records
ReplUpdate	Long integer	Amount of updated and replicated records
ReplDelete	Long integer	Amount of deleted and replicated records
ReplInsert	Long integer	Amount of inserted and replicated records
TotalSelect	Long integer	Total amount of selected records
TotalRead	Long integer	Total amount of read records
TotalReadTime	Long integer	Total data read time (millisecond)
TotalWriteTime	Long integer	Total data write time (millisecond)

Field Name	Type	Description
ActivateTimestamp	Timeline	The time of starting data node
UserCPU	Float	User CPU (second)
SysCPU	Float	System CPU (second)
ErrNodes.NodeName	String	Exception node name ( hostname +port ) Important : this field only display in coord node and existing exception node
ErrNodes.Flag	Int	Error code this field only display in coord node and existing exception node

### Sample

```
> db. snapshot (SDB_SNAP_DATABASE)
{
  "NodeName": "vmsvr2-suse-x64:11820",
  "HostName": "vmsvr2-suse-x64",
  "ServiceName": "11820",
  "GroupName": "datagroup1",
  "IsPrimary": false,
  "ServiceStatus": true,
  "CurrentLSN": {
    "Offset": 208904,
    "Version": 1
  },
  "NodeID": [
    1000,
    1000
  ],
  "Version": {
    "Major": 1,
    "Minor": 3,
    "Release": 8132,
    "Build": "2013-09-26-17.57.31"
  },
  "CurrentActiveSessions": 10,
  "CurrentIdleSessions": 0,
  "CurrentSystemSessions": 8,
  "CurrentContexts": 1,
  "ReceivedEvents": 37,
  "Role": "data",
  "Disk": {
    "DatabasePath": "/opt/sequoiadb/database/data/11820",
    "LoadPercent": 78,
    "TotalSpace": 40704466944,
    "FreeSpace": 8830676992
  },
  "TotalNumConnects": 1,
  "TotalDataRead": 6337,
  "TotalIndexRead": 45,
  "TotalDataWrite": 2112,
  "TotalIndexWrite": 4260,
  "TotalUpdate": 0,
  "TotalDelete": 16,
  "TotalInsert": 2096,
  "ReplUpdate": 0,
  "ReplDelete": 16,
  "ReplInsert": 2096,
  "TotalSelect": 47,
  "TotalRead": 6337,
  "TotalReadTime": 0,
  "TotalWriteTime": 0,
}
```

```

"ActivateTimestamp": "2013-09-27-13. 28. 34. 501403",
"UserCPU": "77. 380000",
"SysCPU": "49. 870000"
}

```

## Operating System Snapshot

### Description

Operating system snapshot contains main status and performance monitoring parameter of operating system of current database node. It returns one record.

### Sign

SDB\_SNAP\_SYSTEM

### Proper nodes

All Node

### Field Information

Field Name	Type	Description
HostName	String	The host name of physical node of database node.
ServiceName	String	Service name specified by svcname. HostName and service name are the sign of a logical node.
NodeName	String	Node name, in the format of <HostName>:<ServiceName>
GroupName	String	The name of replset of the logical node. There is no this field under independent mode.
IsPrimary	Bool	IsPrimary shows whether the node is a main node. There is no this field under independent mode.
ServiceStatus	Bool	ServiceStatus represents the status of whether it is available to serve. For example, <a href="#">Full Sync</a> will make it false.
CurrentLSN.Offset	Long Integer	Excursion of current LSN
CurrentLSN.Version	Integer	Verssion number of the current LSN Ipimportant : this field and above all fields only display in data node and catalog node ,not in coord node.
CPU.User	Float Integer	Total user CPU time (seconds) cost after OS is started.
CPU.Sys	Float Integer	Total system CPU time (seconds) cost after OS is started.
CPU.Idle	Float Integer	Total free CPU time (seconds) cost after OS is started.
CPU.Other	Float Integer	Total other CPU time (seconds) cost after OS is started.
Memory.LoadPercent	Integer	Percentage of used memory of the current operating system (including file system cache) Ipimportant : this field only display in data node
Memory.TotalRAM	Long In teger	Total memory space of the current operating system (byte)



Field Name	Type	Description
Memory.FreeRAM	Long Integer	Free memory space of the current operating system (byte)
Memory.TotalSwap	Long Integer	Total swap space of the current operating system (byte)
Memory.FreeSwap	Long Integer	Free swap space of the current operating system (byte)
Memory.TotalVirtual	Long Integer	Total virtual space of the current operating system (byte)
Memory.FreeVirtual	Long Integer	Free virtual space of the current operating system (byte)
Disk.DatabasePath	String	Database Path Important : this field only display in data node
Disk.LoadPercent	Integer	The percentage of space of file system of data path . Important : this field only display in data node
Disk.TotalSpace	Long Integer	Total space in database path (byte)
Disk.FreeSpace	Long Integer	Free space in database path (byte)
ErrNodes.NodeName	String	Exception node name ( hostname +port ) Important : this field only display in coord node and existing exception node
ErrNodes.Flag	Int	Error code Important : this field only display in coord node and existing exception node

### Sample

```
> db. snapshot (SDB_SNAP_SYSTEM)
{
  "NodeName": "vmsvr2-suse-x64:11820",
  "HostName": "vmsvr2-suse-x64",
  "ServiceName": "11820",
  "GroupName": "datagroup1",
  "IsPrimary": false,
  "ServiceStatus": true,
  "CurrentLSN": {
    "Offset": 3764,
    "Version": 1
  },
  "NodeID": [
    1000,
    1000
  ],
  "CPU": {
    "User": 3947.31,
    "Sys": 715.11,
    "Idle": 331196.41,
    "Other": 771.14
  },
  "Memory": {
    "LoadPercent": 95,
    "TotalRAM": 4155072512,
    "FreeRAM": 202219520,
    "TotalSwap": 2153771008,
    "FreeSwap": 2137071616,
    "TotalVirtual": 6308843520,
    "FreeVirtual": 2339291136
  }
},
```

```

    "Disk": {
      "DatabasePath": "/opt/sequoiadb/database/data/11820",
      "LoadPercent": 78,
      "TotalSpace": 40704466944,
      "FreeSpace": 8615747584
    }
  }
}

```

## Catalog Snapshot

### Description

Catalog Snapshot all the collections listed in the current database catalog information, each set a record.

### Sign

SDB\_SNAP\_CATALOG

### Proper Node

Coord node

### Field Information

Field Name	Type	Description
Name	String	Full name of collection
EnsureShardingIndex	Bool	Whether to automatically create an index for the partitioning key field.
ReplSize	Integer	Modify operation performed when the number of copies to be synchronized. When performing an update, insert, delete records, and so, only when the specified number of copies of the node to complete the operation returns only when the operation result.
ShardingKey	Object	Partitioning key definitions
ShardingType	String	Data partition type: range : Data by partition key range partitioning storage. hash : Data according to a hash partitioning key to partition storage.
Version	Integer	Set the version number, the collection's metadata when performing a modify operation increments the version number (for example, data segmentation)
CataInfo. GroupID	Integer	Partition group ID
CataInfo. GroupName	String	Partition group name
CataInfo. LowBound	Object	Data fragmentation range limit
CataInfo. UpBound	Object	Data slice interval lower limit

### Sample

```

>db.snapshot (SDB_SNAP_CATALOG)
{
  "_id": {
    "$oid": "5247a2bc60080822db1cfba2"
  },
  "Name": "foo.bar",

```

```

"Version": 1,
"ReplSize": 1,
"ShardingKey": {
  "age": 1
},
"EnsureShardingIndex": true,
"ShardingType": "range",
"CataInfo": [
  {
    "GroupID": 1000,
    "GroupName": "datagroup1",
    "LowBound": {
      "": {
        "$minKey": 1
      }
    },
    "UpBound": {
      "": {
        "$maxKey": 1
      }
    }
  }
]
}

```

## List

In this section, the following lists will be introduced:

[Contexts list](#)

[Current session contexts list](#)

[Session list](#)

[Current session list](#)

[Collection list](#)

[Collection space list](#)

[Storage units list](#)

[Replset list](#)

Context List

## Description

Context list contains all the contexts corresponding to all the sessions in current data node.

A session is a record. If a session contains one or more contexts, the Contextd array field generate a for each of the contexts.



Note: The manipulation of list will generate a context, so the result set will contains at least one context of current list.

## Sign

SDB\_LIST\_CONTEXTS

## Proper Node

data node, catalog node

## Field Information

Field Name	Type	Description
SessionID	Long integer	Session ID
Contexts	Long integer array	Context ID array. It is a list of all the contexts of the session.

## Sample

```
> db.list(SDB_LIST_CONTEXTS)
{
  "SessionID": 21,
  "Contexts": [
    182
  ]
}
```

## Current Contexts List

## Description

Current contexts list lists the session contexts corresponding to current connections in data nodes.

It returns one record. In a record, Contexts array field contains all the contexts in current session.



Note: The manipulation of list will generate a context, so the result set of it at least contains one context.

## Sign

SDB\_LIST\_CONTEXTS\_CURRENT

## Proper Node

data node, catalog node

## Field Information

Field Name	Type	Description
SessionID	Long integer	Session ID
Contexts	Long integer array	Context ID array. It is a list of all the contexts of the session.

## Sample

```
> db.list(SDB_LIST_CONTEXTS_CURRENT)
{
  "SessionID": 21,
  "Contexts": [
    183
  ]
}
```

## Session List

## Description

Session list contains all the sessions between users and system on current database node. Each session is recorded as one record.

Sign

SDB\_LIST\_SESSIONS

Proper Node

data node,catalog node

#### Field Information

Field Name	Type	Description
SessionID	Integer or long integer	Session ID
TID	Integer	System thread ID corresponding to the session
Status	String	Session status: <ul style="list-style-type: none"> <li>• Creating : creating status</li> <li>• Running : running status</li> <li>• Waiting : waiting status</li> <li>• Idle : idle status of thread pool</li> <li>• Destroying : destroying status</li> </ul>
Type	String	<a href="#">EDU type</a>
Name	String	EDU name. Generally, it is null.

#### Sample

```
> db.list(SDB_LIST_SESSIONS)
{
  "SessionID": 1,
  "TID": 6168,
  "Status": "Running",
  "Type": "TCPListener",
  "Name": ""
}
{
  "SessionID": 2,
  "TID": 6169,
  "Status": "Running",
  "Type": "HTTPListener",
  "Name": ""
}
...
{
  "SessionID": 21,
  "TID": 6691,
  "Status": "Running",
  "Type": "Agent",
  "Name": "192.168.20.101:52741"
}
```

#### Current Session List

##### Description

Current session list contains the current sessions on current database node. Each session is recorded as one record.

Sign

SDB\_LIST\_SESSIONS\_CURRENT

### Proper Node

data node,catalog node

### Field Information

Field Name	Type	Description
SessionID	Integer or long integer	Session ID
TID	Integer	System thread ID corresponding to the session
Status	String	Session status: <ul style="list-style-type: none"> <li>Creating : creating status</li> <li>Running : running status</li> <li>Waiting : waiting status</li> <li>Idle : idle status of thread pool</li> <li>Destroying : destroying status</li> </ul>
Type	String	<a href="#">EDU type</a>
Name	String	EDU name. Generally, it is null.

### Sample

```
> db.list(SDB_LIST_SESSIONS_CURRENT)
{
  "SessionID": 21,
  "TID": 6691,
  "Status": "Running",
  "Type": "Agent",
  "Name": "192.168.20.101:52741"
}
```

### Collections List

#### Description

Collections list contains all the non-temporary collections in current data nodes (It contains user collection in coord node). Each collection is recorded as a record.

#### Sign

SDB\_LIST\_COLLECTIONS

### Proper Node

All nodes

### Field Information

Since information stored in data nodes and catalog node is not the same, collections list will return different structures according to the kind of the node:

#### Other node field information

Field Name	Type	Description
Name	String	Full name of collection
Details.ID	Integer	Collection ID. Range from 0 to 4095. Unique in collection space.
Details.LogicalID	Integer	Collection logical ID

Field Name	Type	Description
Details.Sequence	Integer	Sequence number
Details.Indexes	Integer	The amount of indexes in the collection
Details.Status	String	The current status of the collection <ul style="list-style-type: none"> <li>• Free</li> <li>• Normal</li> <li>• Dropped</li> <li>• Offline Reorg Shadow Copy Phase</li> <li>• Offline Reorg Truncate Phase</li> <li>• Offline Reorg Copy Back Phase</li> <li>• Offline Reorg Rebuild Phase</li> </ul>

#### Coord node field information

Field Name	Type	Description
Name	String	Full name of collection

#### Sample of other nodes:

```
> db.list(SDB_LIST_COLLECTIONS)
{
  "Name": "foo.test",
  "Details": [
    {
      "ID": 0,
      "Logical ID": 0,
      "Sequence": 1,
      "Indexes": 2,
      "Status": "Normal"
    }
  ]
}
```

#### Coord node Sample:

```
>db.list(SDB_LIST_COLLECTIONS)
{
  "Name": "foo.bar"
}
```

#### Collection Space List

##### Description

Collection space list contains all the collection space in the current data node. Each collection space is recorded as one record.

##### Sign

SDB\_LIST\_COLLECTIONSPACES

##### Proper Node

All nodes

### Field Information

Since information stored in data nodes and catalog node is not the same, collection space list will return different structures according to the kind of the node:

#### Other node field information

Field Name	Type	Description
Name	String	Collection space name
Collection	String array	All the collections in collection space
PageSize	Integer	Size of data page in collection space

#### Coord node field information

Field Name	Type	Description
Name	String	Full name of collection

#### Other nodes sample

```
> db.list(SDB_LIST_COLLECTIONSPACES)
{
  "Collection": [
    {
      "Name": "test"
    }
  ],
  "Name": "foo",
  "PageSize": 4096
}
```

#### Coord node Sample:

```
>db.list(SDB_LIST_COLLECTIONSPACES)
{
  "Name": "foo"
}
```

### Storage Units List

#### Description

Storage units list contains information of all the storage units on the current database node.

#### Sign

SDB\_LIST\_STORAGEUNITS

#### Proper Node

Data node,catalog node

### Field Information

Field Name	Type	Description
Name	String	Collection space name.
ID	Integer	Collection space ID
LogicalID	String	Collection space logical ID, ascending order
PageSize	Integer	Size of data page in collection space
Sequence	Integer	Sequence. In current verssion, it is "1".



Field Name	Type	Description
NumCollections	Integer	The amount of collections in collection space.
CollectionHWM	Integer	Collection high water marks. Generally, it is the amount of collections which has been generated in the collection space (including deleted collection).
Size	Long Integer	Size of storage unit (byte).

### Sample

```
> db.list(SDB_LIST_STORAGEUNITS)
{
  "Name": "testCS",
  "ID": 4095,
  "LogicalID": 0,
  "PageSize": 4096,
  "Sequence": 1,
  "NumCollections": 1,
  "CollectionHWM": 1,
  "Size": 172032000
}
{
  "Name": "foo",
  "ID": 4094,
  "LogicalID": 1,
  "PageSize": 4096,
  "Sequence": 1,
  "NumCollections": 2,
  "CollectionHWM": 3,
  "Size": 172032000
}
```

### ReplicaGroup list

#### Description

ReplicaGroup list contains information of all shards in th current cluster.

#### Sign

SDB\_LIST\_GROUPS

#### Proper node

coord node

#### Field Information

Field Name	Type	Description
Group.dbpath	String	The path of data file of nodes in replica group
Group.HostName	String	Host name of node in replica group
Group.Service.Type	Integer	Service type of node in replica group <ul style="list-style-type: none"> <li>0 : direct-connection service, corresponding to database parameter svcname</li> <li>1 : replication service, corresponding to database parameter replname</li> </ul>

Field Name	Type	Description
		<ul style="list-style-type: none"> <li>2 : shard service, corresponding to database parameter shardname</li> <li>3 : catalog service, corresponding to database parameter catalogname</li> </ul>
Group.Service.Name	String	Service name of node in replica group. Service name may be port number or host name in service file.
Group.NodeID	Integer	Node ID in replica group
GroupID	Integer	Replica group ID
GroupName	String	Replica group name
PrimaryNode	Integer	Master nodeID
Role	Integer	Replica group role.It can be: <ul style="list-style-type: none"> <li>0 : data node</li> <li>2 : catalog node</li> </ul>
Status	String	Status of replica group <ul style="list-style-type: none"> <li>1 : active repleset</li> <li>0 : unactive repleset</li> <li>does not exist : unactive repleset</li> </ul>
Version	String	

### Sample

```
> db.list(SDB-LIST-GROUPS)
{
  "Group": [
    {
      "dbpath": "/home/users/chenzichuan/sequoiadb/cata",
      "HostName": "ubuntu-dev2",
      "Service": [
        {
          "Type": 0,
          "Name": "11800"
        },
        {
          "Type": 1,
          "Name": "11801"
        },
        {
          "Type": 2,
          "Name": "11802"
        },
        {
          "Type": 3,
          "Name": "11803"
        }
      ]
    },
    {
      "NodeID": 1
    }
  ],
  "GroupID": 1,
  "GroupName": "SYSCatalogGroup",
  "PrimaryNode": 1,
  "Role": 2,
  "Status": 1,
}
```

```

"Version": 1,
"_id": {
  "$oid": "51710981d8cb8fbc163d6350"
}
}

```

## Engine Dispatchable Unit

### Concept

Engine Dispatchable Unit is the carrier of tasks in SequoiaDB. Generally, EDU is an independent thread.

Each EDU can be used to execute users' requests or execute system internal maintenance tasks.

EDU is independent to each other. One EDU is charged for one user session. One user session is fixed on one EDU on one data node.

Each EDU has a thread-unique and 64-bit integer ID called "EDU ID".

There are two kinds of EDU: user EDU and system EDU, respectively representing threads that executes user task and threads that executes system task.

### User EDU

User EDU is a thread executes user task. It is also called Agent

In SequoiaDB, there are different kinds of agents as follow:

Name	Type	Description
Agent	Agent	Agent thread is charged for requests of svcname service. Generally, the request is directly sent by user.
ShardAgent	Shard agent	Shard agent thread is charged for requests of shardname service. Generally, the request is directly sent from coord node to data node.
CoordAgent	Coord agent	Coord agent thread is charged for request of svcname service. Generally, the request is directly sent from user to coord node and only acts on coord node.
ReplAgent	Replication agent	Replication agent thread is charged for request of replname service. Generally, the request is directly sent from data master node to data slave node and acts on non-coord node.
HTTPAgent	HTTP agent	HTTP agent thread is charged for request of httpname service. Generally, the request is directly sent by user.

### System EDU

System EDU is a kind of threads that maintains data structures in the system and guarantee the consistency of data and configuration. Generally, it is invisible to users.

In SequoiaDB, there are different kinds of EDU. The following EDU are just part of them:

Name	Type	Description
TCPLListener	Service listener	The duty of this thread is to listen to svcname service, and start agent threads.

Name	Type	Description
HTTPListener	HTTP listener	The duty of this thread is to listen to httpname service, and start agent threads.
Cluster	Cluster management	Cluster management thread is used to maintain the infrastructure of clusters, and start ReplReader threads and ShardReader threads.
ReplReader	Replication listener	ReplReader is charged for request of replname service, and start ReplAgent threads.
ShardReader	Shard listener	ShardReader is charged for request of shardname service, and start ShardAgent threads.
LogWriter	Log writer	LogWriter thread is used to write data from log cache to log file.
WindowsListener	Windows event listener	It is owned by Windows environment. It is used to listen to events defined by SequoiaDB.
Task	Background task	It is a kind of threads that cope with background tasks. For example, <a href="#">data split</a>
CatalogMC	Catalog main controller	CatalogMC thread is used to receive and send requested sent to catalog nodes.
CatalogNM	Catalog node maintainer	CatalogNM thread is used to cope with requests relative to cluster information within catalog nodes.
CatalogManager	Catalog manager	CatalogManager thread is used to cope with requests relative to metadata information within catalog nodes.
CatalogNetwork	Catalog network listener	CatalogNetwork thread is used to listen to requests of catalogname service in coord network.
CoordNetwork	Coord network listner	CoordNetwork thread is used to listen to shard requests.

## Monitorinf

Users can use [session snapshot](#) to monitor system and user EDU.

## Log

### Sync Log

#### Sync-Log

#### Log Files

In SequoiaDB, it with log for data synchronization among data nodes, the log files is in the directory replicalog. The size and number of log file can be configured by setting logfilesz and logfilenum, default is 64M and 20. It can't be modified if the params has entried into force. (if you want to modify, you must delete all the log files off-line, reconfigure the params and restart SequoiaDB. But this will lead to full-sync.)

#### Sync

All slave nodes in data group will regularly packaged download the logs of other nodes to local for log replaying. Sync source is not limited to the master node. As we hope the gap of data version in all nodes is in the range of a small window. When in the range of window, all the slave nodes synchronize

to master node. But when the data version of some nodes is too large differ from master node, they will select other slave nodes to synchronize. And if have conflicted versions, be based on the current data version of master node. If the confliction can't be solved, then enter into full-sync. No master node, no sync.

### Full-sync

Causes of triggering full-sync:

1. downtime and restart.
2. data version of some nodes is too large differ from other nodes.
3. the data is inconsistent and can't be repaired.



Note: After restarting in normal, if the data version is still in the range of synchronization, then it will not trigger full-sync.

the node which experienced full-sync will clear all of local data and logs, and copy all of data in another node (not limit master node) in the group to local, as well as the changed data in sync source. The node in the time of full-sync does not provide services to external, and no master node, no sync. Full-sync will greatly affect the performance of data group, and even lead to lower the synchronous performance of other slave nodes. So adding sharding or increasing log size to avoid full-sync for proposing.

## DataBase Tool

contents :

[Data Migration — Import](#)

[Data Migration — Export](#)

[Database detecting tool — sdbinspt](#)

Data Migration — Import

### sdbimprt

sdbimprt is a import tool in SequoiaDB database, it can achieve from a JSON format or a CSV format data stored file import into SequoiaDB database.

The record of JSON format must meet the json definition: Left and right curly braces ({}), as a record delimiter; and the string data type must be contained in double quotation marks; and escape character is defined as the backslash (\).

CSV is called for short Comma-Separated Value, each record is separate by 0x0D in default, and fields separated by commas. Users can specify the delimiter between fields, as well as the records. Delimiters defined (only support ASCII characters):

usage	default	be equipped
string delimiter	"	yes
character delimiter	,	yes
record delimiter	'\n'(0x0D)	yes

### Options

Param	Description
--help, -h	return the basic help and usage information.
--version	returns version information.
--hostname, -s	import data from SequoiaDB of the specified hostname. In default, the sdbimprt try to connect to local hostname.

Param	Description
--svcname,-p	specify port.In default,the sdbimprt try to connect to the host which port number is 11810.
--user,-u	database username.
--password,-w	database password.
--delchar,-a	specify the character delimiter." " " " is default,"csv" format effective.
--delfield,-e	specify the field delimiter. "," is default,"csv" format effective.
--delrecord,-r	specify the record delimiter."Wn" is default,"csv" format effective.
--csname,-c	specify the collectionspace name of exported data.
--cname,-l	specify the collection name of exported data.
--insertnum,-n	Batch import records, if you set one, use common way to import, if more than one, use batch mode ,the setting range is 1-100000.
--file	specify the name of file want to import.
--type	specify the format of import data.It can be CSV or JSON.
--fields	Specify import data field names."csv" format effective.
--headerline	Specifies whether the first row of imported data as field names."false" is default,"csv" format effective.
--sparse	Specify the import data, automatically add the field name."ture" is default,"csv" format effective.
--extra	Specify the import data, automatically add value."false" is default,"csv" format effective.
--linepriority	Current delimiter default priority is: record delimiter, character delimiter, field delimiter, the default value is ture; If set to false, then the priority of the separator: character deliiter, record delimiter, field delimiter.
--force	If the data has not utf8 data, forced to import data, default false.
--errorstop	If you encounter an error stop, default false.



Note: Linepriority parameters need to be particularly concerned, if set properly, it may fail to import the data. When the record contains "record separator" and linepriority is true, the tool will give priority in accordance with the "record separator" resolution, which led to the import fails. For example: If the record is {"name":"MikeWn"}, the false linepriority mode should be selected.

## Usage

In the following sample,sdbimprt will import data into local database which port is 11810, and collectionspace name is foo,collection name is bar,and import type is csv,import file is test.csv.

```
sdbimprt -s localhost -p 11810 --type csv --file test.csv -c foo -l bar
```

The data type is optional.If the data type is not specified by the user; it will be automatically detected and determined in the standard shown in the above table.

## Usage

In the following example, sdbimprt imports the data into corresponding collection space "foo" and collection "bar" of the local database which connected with the port 11810, with an import type of cvs and import file of test.csv.

```
sdbimprt -s localhost -p 11810 --type csv --file test.csv -c foo -l bar --fields 'name string default "Jack", age int default 18, phone string'
```



Note: The field can be either specified in the command line or in the first line of the import file. If --fields is specified and --headerline is set to true in the command line, the import tool will use the command line's specified field in the first priority and skip the first line of the import file.

- Example 1: The import file type is csv, and the file name is test.csv. Import the data into collection "bar" of collection space "foo".

The following words are the content of the import file:

```
"Jack", 18, "China"
"Mike", 20, "USA"
```

import command:

```
sdbimprt -s localhost -p 11810 --type csv --file test.csv -c foo -l bar --fields 'name
string default "Anonymous", age int, country'
```

- Example 2: The import type is csv, and the file name is test.csv. Import the data into collection "bar" of collection space "foo".

The following words are the content of the import file:

```
name, age, country
"Jack", 18, "China"
"Mike", 20, "USA"
```

import command:

```
sdbimprt -s localhost -p 11810 --type csv --file test.csv -c foo -l bar --fields 'name
string default "Anonymous", age int, country' --headerline true
```



Note:

Because the fields are specified in the first line of the file in Example 2. If the fields need to be specified again, and set "--headerline true" in the command line, and set "-fields 'name string default 'Anonymous', age int, country'", then the import tool will use the field names in the command line and skip the first line of the import file.

Data types automatically detected by sdbimprt:

CSV data	Data types automatically detected by sdbimprt	Actual data
123	number	123
123.1	number	123.1
+123	number	123
-123	number	-123
2E+2	number	200
-2E+2	number	-200
0123	number	123
"123"	string	123
123a	string	123a
"ab""c"	string	ab"c
"{a:1}"	string	{a:1}
"true"	string	true
"false"	string	false
"null"	string	null
true	boolean	true
false	boolean	false
null	null	null

Data types and default values can be specified since version 1.8.

Syntax: field [type] [default <default value >]

Supported data types: int(integer), long, bool(boolean), double, string, null

Supported special data types: timestamp, date

Example:

```
name string default "Jack", age int default 18, phone string
```

Data and Specified types:

Specified types	Supported data type	Note
int(integer)	int, string, bool	bool type, true converts to 1, false converts to 0
long	long, string, int, bool	bool type, true converts to 1, false converts to 0
bool(boolean)	bool, string, int, long	int and long type, 0 converts to false, any number other than 0 converts to true
double	double, string, int	
string	int, long, bool, double, string, timestamp, date, null	
timestamp	string, long	string type, like "2014-01-01-10.30.00.000000", "1388543400000"; long type, like 1388543400000, both types are correct, the values are in the milliseconds
date	string, long	string type, like "2014-01-01", "1388543400"; long type, like 1388543400, both types are correct, the values are in seconds
null		if the data type is specified as null, then the data will be a null no matters what it really is.

Data type priority:

1. When data types are not specified (special data type are not supported): data types will be automatically detected, the priority is in the order of: null > bool > int > double > long > string
2. When data types are specified: Priority is in the order of: Specified data type > supported data type > null
3. When data types and default values are specified: Priority is in the order of: Specified data type > supported data type > default value > null

Data Migration — Export

sdbexprt

sdbexprt is a practical tool, it can export a JSON format or a CSV format data stored file from SequoiaDB database .

Options

Param	Description
--help,-h	return the basic help and usage information.
--version	returns version information.
--hostname,-s	export data from Sequoiadb of the specified hostname. In default, the sdbexprt try to connect to local hostname.
--svcname,-p	specify port. In default, the sdbexprt try to connect to the host which port number is 11810.
--user,-u	database username.
--password,-w	database password.
--delchar,-a	specify the character separator. The default is ",", "csv" formats.
--delfield,-e	specify the field delimiter. ", " is default, "csv" formats.



Param	Description
--delrecord,-r	specify the record delimiter."Wn" is default.
--csname,-c	specify the collectionspace name of exported data.
--cname,-l	specify the collectionspace of exported data.
--fields	specify one or more field names to export,separated by commas(,).
--included	Specify whether to export to csv first line of field names, default true, csv format is valid.
--file	specify the name of file want to export.
--type	specify the format of export data.It can be CSV or JSON.
--errorstop	If you encounter an error stop, default false.

## Usage

In the following sample,sdbexport will export data from the local database which port is 11810, and collectionspace name is foo,collection name is bar,and export type is csv,export file name is contact,export field are field1 and field2.

```
sdbexprt -s localhost -p 11810 --type csv --file contace --fields field1,field2 --c foo -l bar
```

## Database detecting tool — sdbinspt

sdbinspt is a data file detection tool in SequoiaDB , it can detect the structure of data file true or false and present the result report.

## Authorization

The user must have read permission for the data and index in database when running sdbinspt command.

## Required connection

None

## Options

Params	Description
--help, -h	return the basic help and usage information
--dbpath, -d	specify the path of data file, If not be specified, the default path is current path.
--output, -o	specify the output file,If not be specified, default is screen output
--verbose, -v	whether for ASCII text output(true/false),default is true .
--csname, -c	specify the collectionspace name,If not be specified,all collectionspaces.
--cname, -l	specify the collection name,If not be specified,all collections.
--action, -a	specify a action in one of inspect,dump and all , it can't be null. inspect: to check and report any data corruption. dump: to format data page,and output. all: to check data page corruption and formatted output data page.
--dumpdata -t	The set operating data file(true/false),default is false.
--dumpindex, -i	The set operation index file(true/false),false is default.
--pagestart, -s	Specifies the starting data pages,1 is default.
--numpage, -n	Specifies the data pages need to detect or formatted number, when you specify the -s parameter is non negative, the parameters take effect,default is 1

Params	Description
--record, -p	Specifies the display formatted output data or index content, only the default output data or index metadata.

### Usage

When using sdbinspt tool, be sure that the database process has stopped.

The following sample shows sdbinspt will detect in current directory and formatted output the data and indexes in all of collectionspaces and collections to output.txt file .

```
sdbinspt #d . #o output.txt #v true #a all #t true #i true #p true
```

## Cluster Management

---

This section describe some management conceptions and operations about cluster,including create/delete of catalog/data sharding and increasing of coord node.

By understanding these concepts and methodss of operation,to better achieve expand or adjust the cluster.

catalog :

[create master in cluster](#)

[catalog sharding management](#)

[data sharding management](#)

[coord sharding management](#)

[sample](#)

### Add master to cluster

If want to add a new master(Physical Machine or Virtual Machine) to deploy catalog node or data node,then please configure the master system according to the following steps:

- 1.Install the same operating system with other master,and configure IP address.
- 2.According to [System configuration requirement](#) section configure master/kernel param,and add the corresponding relationship between master and IP address to the directory /etc/hosts.
- 3.Modify each cluster master file in the directory /etc/hosts,add the corresponding relationship between new master and IP address to the directory /etc/hosts.
- 4.According to [System configuration requirement](#) section verify the correctness of configuration.
- 5.According to [SequoiaDB system install](#) section install SequoiaDB software.please keep the configuration management service port and the port existing systems consistency.

### Catalog Shardings Management

This section introduce how to add shardings or nodes in catalog shardings.

directoty:

[create new catalog sharding](#)

[create new node in catalog sharding](#)

Create New Catalog Sharding



Note: If creating new nodes relate to create new master,please refer to the section [Add master to cluster](#) to complete the configuration of host name and params about master.

There is only one catalog sharding in the database cluster. So create new catalog sharding have already completed in the time of installed and there is no need execute creating new sharding operation after installing. Please refer to the section [Configuration and startup of cluster mode](#). operation methods:

```
>db.createReplicaCataGroup(<host>,<service>,<dbpath>,[config])
```

This command is to create a new catalog sharding, at the same time create and start a catalog node, amongs:

host : specify the host name of catalog node.

service : specify the service port of catalog node, please ensure this port and the next three ports are unoccupied; for example, if set port 11800, then ensure 11800/11801/11802/11803 are unoccupied.

dbpath : data file path, be used to store catalog data files, please ensure that the data administrator user has write access. (created when installed, default sdbadmin).

config : optional param, be used to configure more detail params. the form is json. param list refer to the section [database configuration](#), for example configure the param of log size {logfilesz:64}.

### Create New Node In Catalog Sharding



Note: If creating new nodes relate to create new master, please refer to the section [Add master to cluster](#) to complete the configuration of host name and params about master.

With the expansion of physical devices in the cluster, it can create more catalog nodes to improve the reliability of catalog service.

operation methods:

```
>var cataRG = db.getRG("SYSCatalog");
>var node1 = cataRG.createNode(<host>,<service>,<dbpath>,[config]);
>node1.start()
```

The first command is used to get catalog sharding, "SYSCatalogGroup" is the name of catalog sharding.

The second command is used to create a new catalog node, amongs:

params:

host : specify the host name of catalog node.

service : specify the service port of catalog node, please ensure this port and the next three ports are unoccupied; for example, if set port 11800, then ensure 11800/11801/11802/11803 are unoccupied.

dbpath : data file path, be used to store catalog data files, please ensure that the data administrator user has write access. (created when installed, default sdbadmin).

config : optional param, be used to configure more detail params. the form is json. param list refer to the section [database configuration](#), for example configure the param of log size {logfilesz:64}.

The third command is used to start the added catalog node.

## Data Sharding Management

This section introduce how to create data shardings and data nodes in the data sharding.

Directory:

[create new data sharding](#)

[create new nodes in data sharding](#)

### Create New Data Sharding



Note: If creating new nodes relate to create new master, please refer to the section [Add master to cluster](#) to complete the configuration of host name and params about master.

It can configure many data sharding in a cluster, the max configurable is 60000. By adding sharding, you can take advantage of the physical device for horizontal expansion. Theoretically, SequoiaDB can do linear horizontal scalability.

operation methods:

```
>var dataRG = db.createRG("datagroup1")
>dataRG.createNode("sdbserver1", 11820, "/opt/sequoiadb/database/data/11820")
>dataRG.start()
```

The first command is used to create data sharding, and what different from catalog sharding is that this operation will not create any data node. The param is the name of data sharding.

The second command is used to create a new node in the data sharding, many times as needed to execute this command to create more data nodes.

amongst:

host : specify the host name of catalog node.

service : specify the service port of catalog node, please ensure this port and the next three ports are unoccupied; for example, if set port 11820, then ensure 11820/11821/11822/11823 are unoccupied.

dbpath : data file path, be used to store catalog data files, please ensure that the data administrator user has write access. (created when installed, default sdbadmin).

config : optional param, be used to configure more detail params. the form is json. param list refer to the section [database configuration](#), for example configure the param of log size {logfilesz:64}.

The third command is used to start data sharding, and it will start all the data nodes and provide services.

### Create New Nodes In Data Sharding



**Note:** If creating new nodes relate to create new master, please refer to the section [Add master to cluster](#) to complete the configuration of host name and params about master.

Some data sharding may set few copies when creating, but with the increase of physical device, it may need to create the number of copies to improve the reliability of data sharding.

operation methods:

```
>var dataRG = db.getRG(<groupname>);
>var node1 = dataRG.createNode(<host>, <service>, <dbpath>, [config]);
>node1.start();
```

The first command is used to get the data sharding, the param of groupname is the name of data sharding.

The second command is used to create a new data node, for the params, please refer to [create new node in catalog sharding](#).

The third command is used to start data nodes.

### Create coord node

When the size of cluster expanding, coord node is also need to added with the expansion of cluster. For advice, one physical node configure one coord node.

1. create directory for the coord node configuration.

```
mkdir -p /opt/sequoiadb/conf/local/11810 ;
```

11810 is the service port of coord node, users can configure with the need.

2. copy the sample configuration file of coord node.

3. modify the configuration file.

```
cp /opt/sequoiadb/conf/samples/sdb.conf.coord /opt/sequoiadb/conf/local/11810/sdb.conf
```

```
vi /opt/sequoiadb/conf/local/11810/sdb.conf
modify content
# database path dbpath=/opt/sequoiadb/database/coord ;
# this param is the path of database,can modify with need,please ensure the path is already
# exists(if not,manually create)
the following line
# catalog addr(hostname1: servicename1,hostname2: servicename2,...)
# catalogaddr=
modify to
# catalog addr(hostname1: servicename1,hostname2: servicename2,...)
catalogaddr=sdbserver1:11803,sdbserver2:11803,sdbserver3:11803 ;
# this param is the service address and port of catalog
```

4. press:wq,save and quit vi.

5. create the path where storage the data file.

```
mkdir -p /opt/sequoiadb/database/coord ;
```

the path is the previous step configured.

6. start catalog node process.

```
/opt/sequoiadb/bin/sdbstart -c /opt/sequoiadb/conf/local/11810/
```

## Sample

Case one:add new master,add a new data duplicate node in current data sharding

- current environment:

Configure items	Configure Situation
master	contain one master: master one:OS is suse 11 SP2 64bit, host name is vmsvr1-suse-x64,IP is 192.168.1.10
catalog sharding group	the group contain one catalog node,service port is 11800
data sharding group	one data sharding group,group name is datagroup1,contain one data node: data node one:service host name is vmsvr1-suse-x64,port is 11820
coord node	one coord node,port is 11810

- expected result after adjustment

Configure items	Configure Situation
master	contain two masters: master one:OS is suse 11 SP2 64bit, host name is vmsvr1-suse-x64,IP is 192.168.1.10 master two:OS is suse 11 SP2 64bit, host name is vmsvr2-suse-x64,IP is 192.168.1.11
catalog sharding	the group contain one catalog node,service port is 11800
data sharding group	one data sharding group,group name is datagroup1,contain two data nodes: data node one:service host name is vmsvr1-suse-x64,port is 11820 data node two:service host name is vmsvr2-suse-x64,port is 11820
coord node	one coord node,port is 11810

## Operation steps

- step one:configure the new added master

- 1.install operation system SUSE 11 SP2 64bit(keep consistent with the original system ).
- 2.login system with root.
- 3.configure IP address:192.168.1.11.
- 4.execute the following command,configuration master is vmsvr2-suse-x64

```
hostname vmsvr2-suse-x64
```

modify the file in /etc/hosts,add two lines:

```
192.168.1.10 vmsvr1-suse-x64
192.168.1.11 vmsvr2-suse-x64
```

press :wq,save and quit.

- 5.execute the following command,to check if the master configuration is work:

```
ping vmsvr1-suse-x64 //check if the comman ping is ok;
ping vmsvr2-suse-x64 //check if the comman ping is ok;
```

- 6.configure the system kernel param:

open the file /etc/profile,add the following lines at the end of the file:

```
ulimit -Sf unlimited //file size limit
ulimit -St unlimited //CPU time limit
ulimit -Sv unlimited //virtual memory limit
ulimit -Sn 64000 //the number of files limit
ulimit -Sm unlimited //memory size limit
ulimit #Su 32000 //the number of threads limit
```

press:wq,save and quit;

execute source /etc/profile , make the configuration work;

- 7.execute the following command,to check if the system kernel param configuration is work:

```
ulimit #a; check if the configuration is right;
```

- step two:configure the master of original cluster

modify the file of original master vmsvr1-suse-x64 in the /etc/hosts,add a new line in the configuration file:

```
192.168.1.11 vmsvr2-suse-x64
```

press :wq save and quit

- step three:install Sequoiadb in the added master  
refer to [SequoiaDB system install](#) section to insatall the software.
- step four:add data nodes to current sharding

1. executed the following command in the name of master vmsvr2-suse-x64,started sequoiadb shell command line:

```
/opt/sequoiadb/bin/sdb
```

2. In the SequoiaDB shell command line,execute the following command to add a copy data node to current data sharding group:

```
> var db = new Sdb("localhost",11810)
> var datarg = db.getRG("datagroup1")
> node=rg.createNode("vmsvr2-suse-x64",11820,"/opt/sequoiadb /database/data/11820")
> node.start()
```

3. In the SequoiaDB shell command line,execute the following command to check the configuration information of the sharding group,you can see a new data node have added to the data group:

```
>db.listReplicaGroups();
...
```

```

{
  "Group": [
    {
      "dbpath": "/opt/sequoiadb/database/data/11820",
      "HostName": " vmsvr1-suse-x64",
      "Service": [
        {
          "Type": 0,
          {
            "Type": 1,
            "Name": "11821"
          },
          {
            "Type": 2,
            "Name": "11822"
          },
          {
            "Type": 3,
            "Name": "11823"
          }
        ],
        "NodeID": 1000
      ],
    },
    {
      "dbpath": "/opt/sequoiadb/database/data/11820",
      "HostName": " vmsvr2-suse-x64",
      "Service": [
        {
          "Type": 0,
          "Name": "11820"
        },
        {
          "Type": 1,
          "Name": "11821"
        },
        {
          "Type": 2,
          "Name": "11822"
        },
        {
          "Type": 3,
          "Name": "11823"
        }
      ],
      "NodeID": 1001
    }
  ],
  "GroupID": 1000,
  "GroupName": "datagroup1",
  "PrimaryNode": 1000,
  "Role": 0,
  "Status": 1,
  "Version": 3,
  "_id": {
    "$oid": "51d673c1fde96799fbac6aad"
  }
}

```

## Operation and Maintain

[Start and Stop the Cluster](#)[Data Backup](#)[Data Recovery](#)[Fault Restoration](#)[Monitor](#)

## Start and stop the cluster

The section describes starting and stopping the cluster.

[Cluster start](#)[Clustrer stop](#)

Cluster start

Restart the operating system

The operating system will automatically start the service starts sdbcm(sequoiadb cluster manager). The service starts, it automatically starts the All physical machines registered in the /opt/sequoiadb/conf/local directory node. Use the command `ps -elf | grep sequoiadb` can see the current node is starting. Booted nodes. After starting the process called: sequoiadb (service name) is starting the process name is generallyly:

```
/opt/sequoiadb/bin/sequoiadb -c /opt/sequoiadb/conf/local/(service name)
```

Manually start a specific node

When a node in the cluster fails, the user can sdb command line using the following steps to start the node. Assuming SequoiaDB the installation path is /opt/sequoiadb

### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

### 2. Get partition group

```
> dataRG = db.getRG ( "<datagroup1>" ) ;
```

### 3. Get the data node

```
> dataNode = dataRG.getNode ( "<hostname1>", "<servicename1>" ) ;
```

### 4. Start Node

```
> dataNode.start() ;
```

Manually start the data set

When a data cluster group is stopped, users can sdb command line using the following steps to start the data set. This action starts the data set of all data nodes.

### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

### 2. Get partition group

```
> dataRG = db.getRG ( "<datagroup1>" ) ;
```

### 3. Start data set

```
> dataRG.start();
```



## Cluster stop

### Manually stop a specific node

Users can sdb command line using the following steps to stop the data nodes.

1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

2. Get partition group

```
> dataRG = db.getRG ( "<datagroup1>" ) ;
```

3. Get the data node

```
> dataNode = dataRG.getNode ( "<hostname1>", "<servicename1>" ) ;
```

4. Stop node

```
> dataNode.stop() ;
```

### Manually stop data set

User can sdb command line using the following steps to stop the data set. This action will stop all data node in the data set.

1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

2. Get partition group

```
> dataRG = db.getRG ( "<datagroup1>" ) ;
```

3. Stop data group

```
> dataRG.stop() ;
```

### Use the kill command to stop the data node

Users can use kill -15 <pid> normal stop data node. The data in this way to stop the node is considered normal stops. Users use kill -9 <pid> forcibly stop data node. The data in this way to stop the node is considered abnormal stop. If the node using the normal boot process, it will be sdbcm process attempts to restart. Starts with the other noed in the current data synchronization group.

## Data backup

The current version, data backup support offline backup, or data backups need to interrupt the insert, update, delete, etc. change operation, only support query operations. Current backup supports two modes: full backup and incremental backup.

- Full backup: Backing up the entire database configuration, data and logs.
- Incremental backup: In the last full backup or incremental backup on the basis of the new log and configuration.

### Offline backup parameter description

Parameter	Explanation
Name	Backup name, the default name places the current time format, such as "2013-11-13-15:00:00".
Description	Backup user description
Path	The backup path is specified, the default for the configuration parameter "bkuppath" in the path specified.

Parameter	Explanation
EnsureInc	Backup, true means that incremental backups, false that full backup, the default is false.
OverWrite	For the same name as the backup to overwrite, true representation covering, false means not overwritten if the same name is an error, lack of true.
GroupName	Backup of the specified group, the default for the whole system backup, when you need to back up multiple groups can be specified as an array type, such as: ["datagroup1","datagroup2"]

### Backup the entire database

#### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost",11810);
```

#### 2. Perform the bacup command

```
> db.backupOffline({Name: "backupName", Description: "backup for all"})
```

### Database backup designated group

#### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost",11810);
```

#### 2. Perform the backup command

```
> db.backupOffline({Name: "backupName", Description: "backup group1", GroupName: "datagroup1"})
```

### Specified node database backup

#### 1. Connected to the specified node

```
$ /opt/sequoiadb/bin/sdb
> var dbdata = new Sdb("hostname1", "servicename1");
```

#### 2. Perform the backup command

```
> dbdata.backupOffline({Name: "backupName", Description: "backup data node"})
```



Note: catalog name of the group is fixed to SYSCatalogGroup.

## View backup information

View backup information manually through the client and see.

### View backup information Parameter Description

Parameter	Description
Name	Backup name, and then view the default directory for all backup information.
Path	Check the backup path is specified, the default for the configuration parameter "bkuppath" in the path specified.
GroupName	Specifies a group of backup information, which defaults to view full system backup information, when you need to view multiple set of backup information can be specified as an array type, such as: ["datagroup1","datagroup2"].

## View full system backup information

### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost",11810);
```

### 2. Do view backup information command

```
>db. listBackup()
{
  "Name": "test_bk",
  "NodeName": "vmsvr2-suse-x64-1:11800",
  "GroupName": "SYSCatalogGroup",
  "EnsureInc": false,
  "BeginLSNOffset": 0,
  "EndLSNOffset": 18744,
  "StartTime": "2013-11-13-16: 06: 31",
  "HasError": false
}
{
  "Name": "test_bk",
  "NodeName": "vmsvr2-suse-x64-1:11820",
  "GroupName": "db1",
  "EnsureInc": false,
  "BeginLSNOffset": 0,
  "EndLSNOffset": 920424,
  "StartTime": "2013-11-13-16: 06: 31",
  "HasError": false
}
```

## View backup information specified name

### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost",11810);
```

### 2. Do view backup information command

```
>db. listBackup({Name: " backup1" })
{
  "Name": "backup1",
  "NodeName": "vmsvr2-suse-x64-1:11820",
  "GroupName": "group1",
  "EnsureInc": false,
  "BeginLSNOffset": 0,
  "EndLSNOffset": 108744,
  "StartTime": "2013-11-13-16: 06: 31",
  "HasError": false
}
```

## Hand-view backup information

Hand-view backup information directly through the terminal login specified machine and into the appropriate backup directory, run "ls -l"

```
use@vmsvr2-suse-x64-1: /opt/sequoiadb/database/11820/bakfile> ls -l
total 37328
-rw-r----- 1 sdbadmin sdbadmin 38157784 Nov 13 16:06 test_bk.1
-rw-r----- 1 sdbadmin sdbadmin 65536 Nov 13 16:06 test_bk.bak
```

## Data Recovery

Use a partition data recovery backup group. Perform data recovery must ensure that the corresponding group has stopped running, data recovery will first clear the original node and log all the data and then restore the data from the backup configuration data and logs.

Data recovery tool Parameter Description

Parameter	Description
-p [--bkpath]	Backup source where the data path.
-i [--increaseid]	Requires several incremental backups to restore to default back to the last time.
-n [--bkname]	Need to restore the backup name.
-a [--action]	Recovery behavior, "restore" indicates that the recovery, "list" indicates view backup information, defaults to "restore".
--isSelf	Whether to restore the node data, the default is "true", when the value is "false", according to the following parameters to restore the data to the specified path:
--dbpath	Must configure the data file directory.
--confpath	Must configure the configuration file path.
--svcname	Must configure the local service name or port.
--indexpath	Index file directory.
--logpath	Log file directory.
--diagpath	Diagnostic log file directory.
--bkuppath	Backup file directory.
--replname	Copy communications service name or port.
--shardname	Shard communications service name or port.
--catalogname	Catalog communications service name or port.
--httpname	REST service name or port.

### Recover Data



**Note:** If a partition group that contains multiple data nodes, you must stop the group and recover each data node. If the data is restored to a non-backup data nodes must use -isSelf false configuration parameters, and set the relevant configuration parameters.

#### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

#### 2. Get partition group

```
> datarg = db.getRG ( "data" ) ;
```

#### 3. Stop partition group

```
> datarg.stop()
```

#### 4. Login via terminal nodes corresponding data partition group, perform data recovery.

```
sdbadmin@vmsvr2-suse-x64-1: /opt/sequoiadb> bin/sdbrestore -p database/11820/bakfile -n
test_bk
Begin to clean dps logs...
Begin to clean dms storages...
Begin to init dps logs...
Begin to restore...
Begin to restore data file: 11820/bakfile/test-bk.1 ...
Begin to restore su: test.1.data ...
```

```

Begin to restore su: test.1.idx ...
Begin to restore dps logs...
*****
Restore succeed!
*****

```

5. Check the file to a data node directory is restored.

```

sdbadmin @vmsvr2-suse-x64-1: / opt/sequoiadb /database/11820> ls -l
total 299156
drwxr-xr-x 2 sdbadmin sdbadmin      4096 Nov 13 16:06 bakfile
drwxr-xr-x 2 sdbadmin sdbadmin      4096 Nov 13 15:48 diaglog
drwxr-xr-x 2 sdbadmin sdbadmin      4096 Nov 13 17:39 replicalog
-rw-r----- 1 sdbadmin sdbadmin 155254784 Nov 13 17:39 test.1.data
-rw-r----- 1 sdbadmin sdbadmin 151060480 Nov 13 17:39 test.1.idx

```

6. Delete the partition groups other data, all data nodes (or all the nodes. Data and. Idx files are copied to the other data node's data directory and index directory, and copy all the logs that node replicalog to other data nodes log directory, or copy the backup file to another data node, and through the restored tool recovers), restart the system.

## Recovery

This section describes the types of node failure recovery.

[Coord node](#)

[Data node](#)

[Catalog node](#)

Coord node

As the coord node user data does not exist, so you can directly after a failure to restart, do not participate in any additional recovery steps.

Data node

Data node fails, restart will automatically detect the database directory. SEQUOIADB\_STARTUP hidden files.

If the file exists then the last execution terminates unexpectedly (eg kill -9). For the unexpected termination of the node, the system will crash recovery into the data node status.

In crash recovery process, the data node in the group with a normal amount of synchronizing all nodes. In this case, the restored node void all the data synchronized to the new data as a reference. Assuming that the node has not been terminated unexpectedly (eg kill -15), then enter the incremental synchronization status. In this case, if the current node other data contained in the oldest log is already higher than the new node to be restored, then enter the full amount of synchronization state, otherwise only the incremental synchronization log.

If the data set of all nodes are terminated unexpectedly, you need to start a node in standalone mode for local recovery. In this mode, the data will be exported and reimported to filter out all the possible data corruption. When one of the local node is restored, it needs to be copied into the data directory for all other data nodes.

Catalog node

Catalog a node failure recovery strategies and the same data nodes. [Click here](#)

## Monitor

This section describes the node and cluster monitoring information

[Monitoring node status](#)

## Monitoring Cluster

### Monitoring node status

Users can use the snapshot monitor the status of each node.

#### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

#### 2. Get partition group

```
datarg = db.getRG ( "<datagroup1>" ) ;
```

#### 3. Get the data node

```
> datanode = datarg.getNode ( "<hostname1>", "<servicename1>" ) ;
```

#### 4. To obtain a snapshot of the node

```
> datanode.connect().snapshot(SDB_SNAP_DATABASE)
```

Snapshot type are divided into:

[SDB\\_SNAP\\_CONTEXTS](#)

[SDB\\_SNAP\\_CONTEXTS\\_CURRENT](#)

[SDB\\_SNAP\\_SESSIONS](#)

[SDB\\_SNAP\\_SESSIONS\\_CURRENT](#)

[SDB\\_SNAP\\_COLLECTIONS](#)

[SDB\\_SNAP\\_COLLECTIONSPACES](#)

[SDB\\_SNAP\\_DATABASE](#)

[SDB\\_SNAP\\_SYSTEM](#)

[SDB\\_SNAP\\_CATALOG](#)

Users can use the shell script monitoring, for example:

```
[sequoiadb@vmsvr1-rhel-x64 sequoiadb]$ cat monitor-insert.sh
#!/bin/bash
-/sequoiadb/bin/sdb "db=new Sdb('localhost', 11810)" > /dev/null
-/sequoiadb/bin/sdb "db.getRG('foo').getNode('vmsvr1-rhel-x64',11820).connect().snapshot(SDB_SNAP_DATABASE)" | grep TotalInsert
-/sequoiadb/bin/sdb "quit"
[sequoiadb@vmsvr1-rhel-x64 sequoiadb]$ ./monitor-insert.sh
"TotalInsert": 0,
```

## Monitoring Cluster

#### 1. Connect to the coord node

```
$ /opt/sequoiadb/bin/sdb
> var db = new Sdb("localhost", 11810) ;
```

#### 2. Get cluster status

```
> db.listReplicaGroups()
{
  "Group": [
    {
      "dbpath": "/home/sequoiadb/sequoiadb/cata",
      "HostName": "vmsvr1-rhel-x64",
      "Service": [
        {
          "Type": 0,
          "Name": "11800"
        }
      ]
    }
  ],
}
```

## System Security

---

### Concept

The permission to log in the system is specified in security module. If a user without correct user name and password cannot visit database.

### Details

1. When the system is started, security function is closed by default. Only when a user is created, security function can be automatically activated.
2. User name and password should be specified when creating a user. User name is unique in the whole system.
3. User name and password should be specified when deleting a user.
4. After security function is activated, user name and password should be specified when visiting database. In one session, identity verification should be fulfilled once. When a new session is started, identity verification should be fulfilled again.
5. Users are equal to each other. There is no super user. A user can delete any users.
6. When all users are deleted, security function is closed.
7. All passwords are transmitted and stored in the format of cipher text.

# Hadoop Integration

---

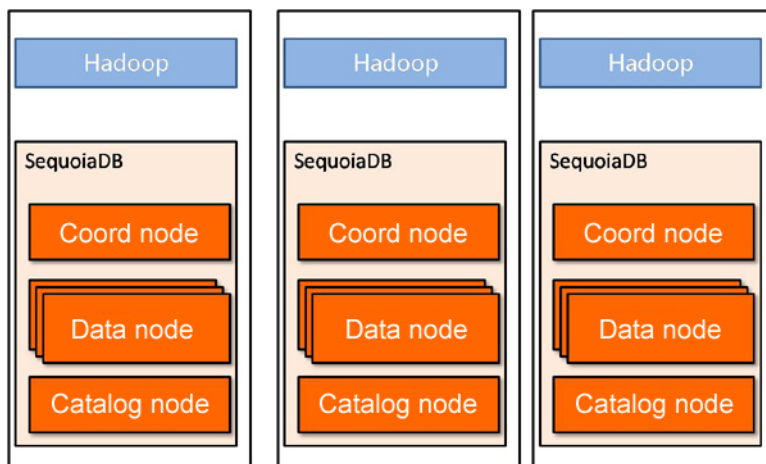
Hadoop integration with related content.

## SequoiaDB with Hadoop deployment

---

SequoiaDB with Hadoop deployment scenario physically simple as shown below, deployment recommendations are as follows:

- SequoiaDB with Hadoop deployed on the same physical device, in order to reduce the network between Hadoop and SequoiaDB data transmission.
- Each physical devices are deployed a coord node and a plurality of data nodes, catalog the node optional equipment at any of the three physical deployment of a catalog node.



## With MapReduce Integration

---

### Build Hadoop Environment

We support both Hadoop 1.x and Hadoop 2.x, please install and configure Hadoop first.

### 2. Configure Docking Environment

hadoop-connector.jar and sequoiadb.jar are used for the docking with MapReduce. These two jar files can be found under the hadoop directory of the SequoiaDB installation directory.

We need to check the classpath of Hadoop first because it may vary in different versions. Enter hadoop classpath, select one directory from the classpath, move haddp-connector.jar and sequoiadb.jar into the directory. Restart the hadoop cluster.

### 3. Write MapReduce

Some important classes in hadoop-connector.jar:

SequoiadbInputFormat: reads data from SequoiaDB

SequoiadbOutputFormat: writes data into SequoiaDB

BSONWritable: BSONObject's wrapper class, it realizes the interfaces of WritableComparable class. Used to serialize BSONObject objects.



## The Configuration of SequoiaDB & MapReduce

Put the configuration file named sequoiadb-hadoop.xml into the root directory of the source code of the project.

sequoiadb.input.url: Specify the URL of the SequoiaDB which acts as an input source, the format is: hostname1:port1, hostname2:port2,

sequoiadb.in.collectionspace: Specify the collection space which acts as an input source.

sequoiadb.in.collection: Specify the collection which acts as an input source.

sequoiadb.output.url: Specify the URL of the SequoiaDB which acts as an output target.

sequoiadb.out.collectionspace: Specify the collection space which acts as an output target.

sequoiadb.out.collection: Specify the collection which acts as an output target.

sequoiadb.out.bulknum: Specify the number of records that are written into SequoiaDB for each time in order to optimize the write performance.

### Examples

- 1. The following codes read and process the data form HDFS files, and then write the result into SequoiaDB.

```
public class HdfsSequoiadbMR {
    static class MobileMapper extends Mapper<LongWritable,Text,Text,IntWritable>{
        private static final IntWritable ONE=new IntWritable(1);
        @Override
        protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
            String valueStr=value.toString();

            String mobile_prefix=valueStr.split(",")[3].substring(0,3);
            context.write(new Text(mobile_prefix), ONE);
        }
    }

    static class MobileReducer extends Reducer<Text, IntWritable, NullWritable,
    BSONWritable>{

        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,Context context)
            throws IOException, InterruptedException {
            Iterator<IntWritable> iterator=values.iterator();
            long sum=0;
            while(iterator.hasNext()) {
                sum+=iterator.next().get();
            }
            BSONObject bson=new BasicBSONObject();
            bson.put("prefix", key.toString());
            bson.put("count", sum);
            context.write(null,new BSONWritable(bson));
        }
    }

    public static void main(String[] args) throws IOException, InterruptedException,
    ClassNotFoundException {
        if(args.length<1) {
            System.out.print("please set input path ");
            System.exit(1);
        }
    }
}
```

```

    }
    Configuration conf=new Configuration();
    conf.addResource("sequoiadb-hadoop.xml"); //load the configuration file
    Job job=Job.getInstance(conf);
    job.setJarByClass(HdfsSequoiadbMR.class);
    job.setJobName("HdfsSequoiadbMR");
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(SequoiadbOutputFormat.class); //the reduce oupput is
written to Sequoiadb
    TextInputFormat.setInputPaths(job, new Path(args[0]));

    job.setMapperClass(MobileMapper.class);
    job.setReducerClass(MobileReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(NullWritable.class);
    job.setOutputValueClass(BSONWritable.class);

    job.waitForCompletion(true);
}
}

```

- 2. Reads and processes the data from Sequoiadb, and then writes the result into HDFS.

```

public class SequoiadbHdfsMR {
    /**
     *
     * @author gaoshengjie
     * read the data, count penple in a province
     */
    static class ProvinceMapper extends Mapper<Object, BSONObject, IntWritable, IntWritable>{
        private static final IntWritable ONE=new IntWritable(1);
        @Override
        protected void map(Object key, BSONObject value, Context context)
            throws IOException, InterruptedException {
            int province=(Integer) value.get("province_code");
            context.write(new IntWritable(province), ONE);
        }
    }

    static class ProvinceReducer extends
Reducer<IntWritable, IntWritable, IntWritable, LongWritable>{

        @Override
        protected void reduce(IntWritable key, Iterable<IntWritable> values,
            Context context)
            throws IOException, InterruptedException {
            Iterator<IntWritable> iterator=values.iterator();
            long sum=0;
            while(iterator.hasNext()) {
                sum+=iterator.next().get();
            }
            context.write(key, new LongWritable(sum));
        }
    }

    public static void main(String[] args) throws IOException, InterruptedException,
ClassNotFoundException {
        if(args.length<1) {
            System.out.print("please set output path ");

```

```

        System.exit(1);
    }
    Configuration conf=new Configuration();
    conf.addResource("sequoiadb-hadoop.xml");
    Job job=Job.getInstance(conf);
    job.setJarByClass(SequoiadbHdfsMR.class);
    job.setJobName("SequoiadbHdfsMR");
    job.setInputFormatClass(SequoiadbInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileOutputFormat.setOutputPath(job, new Path(args[0]+"/result"));

    job.setMapperClass(ProvinceMapper.class);
    job.setReducerClass(ProvinceReducer.class);

    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(IntWritable.class);

    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(LongWritable.class);

    job.waitForCompletion(true);
}
}

```

configuration file:

```

<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property>
    <name>sequoiadb.input.url</name>
    <value>localhost:11810</value>
  </property>
  <property>
    <name>sequoiadb.output.url</name>
    <value>localhost:11810</value>
  </property>
  <property>
    <name>sequoiadb.in.collectionspace</name>
    <value>default</value>
  </property>
  <property>
    <name>sequoiadb.in.collect</name>
    <value>student</value>
  </property>
  <property>
    <name>sequoiadb.out.collectionspace</name>
    <value>default</value>
  </property>
  <property>
    <name>sequoiadb.out.collect</name>
    <value>result</value>
  </property>
  <property>
    <name>sequoiadb.out.bulknum</name>
    <value>10</value>
  </property>
</configuration>

```

## Integration with Hive

- [SequoiaDB list of the supported versions Hive](#)

- [Configuration method](#)

## SequoiaDB list of the supported versions Hive

- Apache Hive 0.12.0
- Apache Hive 0.11.0
- Apache Hive 0.10.0
- CDH-5.0.0-beta-2 Hive 0.12.0



Note: hive-sequoiadb-apache.jar support for the Apache version of Hive Sequoiadb-Hive-Connector.

hive-sequoiadb-cdh-5.0.0-beta-2.jar support for the CDH5.0.0-beta-2 version of Hive-0.12 Sequoiadb-Hive-Connector.

## Configuration method

1. Install and configure Hadoop/Hive environment, start hadoop environment.
2. Copy the sequoiadb installation directory (by default in /opt/sequoiadb) of hadoop/hive-sequoiadb-{version}.jar and java/sequoiadb.jar two files are copied to the hive/lib installation directory.
3. Modify the hive installation directory under bin/hive-site.xml file (if you do not exist, you can copy the \$HIVE\_HOME/conf/hive-default.xml.template to hive-site.xml file), add the following attributes:

```
<property>
  <name>hive. aux. jars. path</name>
  <value>file: //<HIVE-home>/lib/hive-sequoiadb- {version}. jar, file: //<HIVE-HOME>/lib/
sequoiadb. jar</value>
  <description>Sequoiadb store handler jar file</description>
</property>

<property>
  <name> hive. auto. convert. join</name>
  <value>false</value>
</property>
```

## Use

- [Create a table based SequoiaDB](#)
- [Importing data form HDFS files to SequoiaDB table](#)
- [Query Data](#)

### Create a table based SequoiaDB

Start hive shell command line window, execute the following command to create the data table.

```
hive> create external table sdb_tab(id INT, name STRING, value DOUBLE) stored by
"com.sequoiadb.hive.SdbHiveStorageHandler" tblproperties ("sdb.address" = "localhost:11810");
OK
Time taken: 0.386 seconds
```

Of which:

Sdb.address SequoiaDB coord node is used to specify the IP and port, if there are multiple coord node, you can write to multiple, separated by commas.

### Importing data form HDFS files to SequoiaDB table

```
hive> insert overwrite table sdb_tab select * from hdfs_tab;

Total MapReduce jobs = 1
Launching Job 1 out of 1
```

```

Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201310172156-0010, Tracking URL = http://bl465-5:50030/jobdetails.jsp?
jobid=job_201310172156-0010
Kill Command = /opt/hadoop-hive/hadoop-1.2.1/libexec/./bin/hadoop job -kill
job_201310172156-0010
Hadoop job information for Stage-0: number of mappers: 1; number of reducers: 0
2013-10-18 04:44:47,733 Stage-0 map = 0%, reduce = 0%
2013-10-18 04:44:49,763 Stage-0 map = 100%, reduce = 0%, Cumulative CPU 1.85 sec
2013-10-18 04:44:50,777 Stage-0 map = 100%, reduce = 0%, Cumulative CPU 1.85 sec
2013-10-18 04:44:51,795 Stage-0 map = 100%, reduce = 100%, Cumulative CPU 1.85 sec
MapReduce Total cumulative CPU time: 1 seconds 850 msec
Ended Job = job_201310172156-0010
10 Rows loaded to sdb_tab
MapReduce Jobs Launched:
Job 0: Map: 1 Cumulative CPU: 1.85 sec HDFS Read: 2301 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 850 msec
OK
Time taken: 12.201 seconds

```

Description: When you import data into SequoiaDB table, make sure that you have created on HDFS file `hdfs_tab` data sheet, and Load data.

### Query Data

```

hive> select * from new_tab;
OK
0      false    0.0      ALGERIA
1      true     1.0      ARGENTINA
2      true     1.0      BRAZIL
3      true     1.0      CANADA
4      true     4.0      EGYPT
5      false    0.0      ETHIOPIA
6      true     3.0      FRANCE
7      true     3.0      GERMANY
8      true     2.0      INDIA
9      true     2.0      INDONESIA
Time taken: 0.306 seconds, Fetched: 10 row(s)

```

## Integration with Pig

---

No documentation.

# Development Instruction

---

Sequoiadb relative development information

## SequoiaDB shell

---

Catalog

- [Sequoiadb shell Introduction](#)
- [Tips when using shell](#)

### SequoiaDB Shell Introduction

SequoiaDB has a built-in javascript shell, users can communicate with SequoiaDB instance through command lines. It is very useful. Through it, users can manage system, check and run instance or try other things. Shell is a vital tool in SequoiaDB.

Run shell

Run sequoiadb(./sequoiadb) and start shell :

```
$ ./sdb
Welcome to SequoiaDB shell!
help() for help, Ctrl+c or quit to exit
>
```

Shell will automatically connect to SequoiaDB when SequoiaDB starts, so users should start SequoiaDB before using shell.

Shell is a javascript resolver with complete functions, which can run any javascript program. For example:

```
> y=200
200
> y/20
10
>
```

Users can also use javascript stdlib:

```
> new Date("2013/04/17");
Wed Apr 17 2013 00:00:00 GMT+0800 (CST)
> "hello,world".replace("world", "SequoiaDB")
hello, SequoiaDB
>
```

Uses can define and invoke javascript function:

```
> function sdb(n) {
...   if(n<=1) return 1;
...   else return n*sdb(n-1);
... }
> sdb(4);
24
>
```



**Note:** We can use multi-line commands. Shell will check out whether javascript statement is complete. If it is not complete, users can continue to input next line.

## SequoiaDB Client

Shell can run any javascript program. But the particular feature of shell is that is a unique SequoiaDB client. When the system is started, with the command "db=new Sdb("localhost",11810)", users can connect to local database in sequoiaDB server and assign it to the global variable "db". This variable is the a main entrance to visit SequoiaDB in shell.

Shell also has some non-javascript commands. For example:

```
>db
localhost:11810
>db.create("foo");//create colleciton space
localhost:11810.foo
```

Users can visit the collection space in it through the variable "db". "db.foo" returns the collection space "foo" in current database. Users can visit collection in it like "db.foo.bar" which returns the collection "bar" in collection "foo". Since users can visit collections in shell, they can execute almost all manipulations in database.

## Tips when using shell

SequoiaDB shell has built-in help documents. It can be accessed by the help() command. For more details on each method, please refer to the SeuoiaDB JavaScript Method List in the Reference section of SequoiaDB Information Center.

- Help

See Use description:

```
> help()
  var db = new Sdb()                connect to database use default host 'localhost'
and default port 11810
  var db = new Sdb('localhost',11810) connect to database use specified host and port
  var db = new Sdb('ubuntu',11810,'','') connect to database with username and password
  help(<method>)                    help on specified method, e.g. help('createCS')
  db.help()                         help on db methods
  db.cs.help()                      help on collection space cs
  db.cs.cl                          help on collection cl on collection space cs
  db.cs.cl.help()                   help on collection cl
  db.cs.cl.find()                   list all records
  db.cs.cl.find({a:1})               list records where a=1
  db.cs.cl.find().help()             help on find methods
  db.cs.cl.count().help()           help on count methods
  print(x), println(x)              print out x
  traceFmt(<type>,<in>,<out>)         format trace input(in) to output(out) by type
  getErr(ret)                       print error description for return code
  clear                             clear the terminal screen
  history -c                         clear the history
  quit                              exit
Takes 0.2993s.
```



### Note:

SequoiaDB shell mainly includes 7 levels of operations: database (db), collection space (cs), collection (cl), cursor (cur), replica group (rg), node (nd), and domain (dm). Users need to understand the relationship between different levels. The operations of each level have its own help command:

- Database Help

Database level's main operations are user group management, collection space, replica group, domain, snapshot, storage process, backup, transaction, SQL, and error tracing.

Assume the database has been connected, and there is a javascript object named db which is an instance of the database.

Check all the methods of a database:

```
db.help()
```

Check a specific method of a database:

```
db.help("method")
```

- **CollectionSpace Help**

Collection Space level's main operation is the management on the collections.

Assume there is a collection space named "foo".

Check all the methods of a collection space:

```
db.foo.help()
```

Check a specific method of a collection space:

```
db.foo.help("method")
```

- **Collection Help**

Collection level's main operations are CRUD, index management, data sharding, and management on the vertical partition table.

Assume "foo" hypothesis in the set space exists in name as "bar" collection.

Check all the methods of a collection:

```
db.foo.bar.help()
```

Check a specific method of a collection:

```
db.foo.bar.help("method")
```

- **Cursor Help**

Cursor level's main operations are on the returned data.

All the data are returned in the form of cursor when interacting with SequoiaDB engine in the shell command. For example, when execute a query using `db.foo.bar.find()` method, a cursor object will be returned with all the query results in it.

```
db.foo.bar.find()
```

or

```
var cur = db.foo.bar.find()
```

The first one directly display all the results on the screen, the second one put the results into a cursor.

Check all the methods of a cursor:

```
db.foo.bar.find().help()
```

or

```
cur.help()
```

Check a specific method of a cursor:

```
db.foo.bar.find().help("method")
```

or

```
cur.help("method")
```

The methods similar to `find()` that also return cursor are `list()`, `snapshot()`, etc.

- **Replica Group Help**

Replica group level's main operation is the management on the data node.

Assume there is a replica group named "group1" in the database, a javascript object named `rg` which is a instance of a replica group is obtained by using "`var rg = db.getRG("group1")`" command.



Check all the methods of a replica group:

```
rg.help()
```

Check a specific method of a replica group:

```
rg.help("method")
```

- **Node Help**

Node level's main operation is the obtaining of the status information of data nodes.

Assume creating a data node in the replica group named "group1" using "var rn = rg.createNode("ubuntu-dev1", 51000, "/opt/sequoiadb/database/data/51000")", a javascript object named rn which is an instance of the data node will be obtained.

Check all the methods of a node:

```
rn.help()
```

Check all the methods of a node:

```
rn.help("method")
```

- **Domain Help**

Domain level's main operations are the modification of domains and the obtaining of the domain information.

Assume creating a domain named "domain1" in the database using "var dm = db.createDomain("domain1",["group1","group2"],{AutoSplit:true})" command, a javascript object named dm which is an instance of domain will be obtained.

Check all the methods of a domain:

```
dm.help()
```

Check a specific method of a domain:

```
dm.help("method")
```



**Note:** SequoiaDB's man page was added in Version 1.8. The previous versions do not support the help("method") function. Moreover, please make sure that there is a "troff" file under /opt/sequoiadb/doc/manual directory; otherwise, man page will not be displayed.

Please visit [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language\\_Resources](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources) to check all the automatically generated javascript function API documents provide by shell.

## Basic Manipulation in Shell

---

In shell, there are 4 basic manipulations :create, read, update and delete (CRUD).

- [create](#)
- [read](#)
- [update](#)
- [delete](#)

### Create

In sequoiaDB, "create" means adding new document record into collections. We can add new record into collections through the method "insert". In sequoiaDB, the insert manipulations are as follow:

- If there is no "\_id" field in the document record which is about to be inserted, the client will automatically add "\_id" field and insert an unique value.
- If "\_id" field is specified, the "\_id" field in that collection should be unique, or exception will occur.
- The maximum length of BSON document is 16MB.

- The limits of fields are as follow:

The field "\_id" should be stored as primary key in a collection. The value of it should be unique and unchangable. The type of "\_id" can be ordinary types except for array.

The name of a field should not be null, start with "\$" or contain (.).



Note: In this document, all examples use shell interface in sequoiadb.

insert()

The method `insert()` is used to insert record into insert records into sequoiadb collection. The grammar of it is :

```
db.collectionspace.collection.insert(<doc|docs>, [flag])
```

Insert the first document

If there is no [collection space](#) and [collection](#). Firstly, create collection (For example, "db.createCS("foo")" create collection space "foo") and collection (For example, "db.foo.createCL("bar")" create collection under collection space "bar"), then you are able to insert records.

```
db.foo.bar.insert(
{
  _id: 1,
  name: {first: "Jhon", last: "Black"},
  phone: [1853742XXX, 1802321XXX],
  remark: [
    {
      position: "manager",
      year: 2000
    },
    {
      position: "CEO",
      year: 2012
    }
  ]
})
```

Users can check out whether the insert is successful through the method "find()"

```
db.foo.bar.find()
```

The result is as follow:

```
{
  _id: 1,
  name: {first: "Jhon", last: "Black"},
  phone: [1853742XXX, 1802321XXX],
  remark: [
    {
      position: "manager",
      year: 2000
    },
    {
      position: "CEO",
      year: 2012
    }
  ]
}
```

### unspecified \_id field

If a new document record doesn't contain "\_id" field, the method `insert()` will add "\_id" field into a document and generate an unique "\$oid" value.

```
db.foo.bar.insert({name: "Tom", age: 20})
```

This command insert a new record into collection "bar". In the record, the value of "name" is "Tom". The value of "age" is 20. The "\_id" field is unique:

```
{ "_id": { "$oid": "515152ba49af395200000000" }, "name": "Tom", "age": 20 }
```

### Insert more than one record

If users insert an array document in the method "insert", the method "insert()" will execute batch inserts in a collection.

The following command inserts 2 records into collection "bar". This manipulation demonstrates the feature of dynamic mode in sequoiadb. Although the record with "\_id:20" contains field name "phone", but the other one doesn't contain that, sequoiadb doesn't require all records to contain this field.

```
db.foo.bar.insert([ {name: "Mike", age: 15}, { _id: 20, name: "John", age: 25, phone: 123} ])
```

## Read

Of the basic database operations(i.e.CRUD),read operations are those that retrieve records or documents from a collection in Sequoiadb, and include all operations that return a cursor in response to application request data.

This document describes the syntax and structure of the queries applications use to request data from Sequoiadb and how different factors affect the efficiency of reads.



#### Note:

All of the examples in this document use the sequoiadb shell interface.

### find()

We use `find` method to read records from Sequoiadb. The `find` method is the main method to select records from collections and it returns a cursor which contains a lot of records. Its grammatical structure is as follows:

```
db.collectionspace.collection.find([cond], [sel])
```

The corresponding operation in SQL: the method of `find()` is similar with `select` statement :

.the `[cond]` parameter corresponds to `where` statement

.the `[sel]` parameter corresponds to the list of fields in the result set

there is a record in collection like this :

```
{
  "_id": 1,
  "name": {
    "first" : "Tom",
    "second": "David"
  },
  "age": 23
  "birth": "1990-04-01",
  "phone": [
    10086,
    10010,
    10000
  ],
  "family": [
    {
```

```

    "Dad": "Kobe",
    "phone": 139123456
  },
  {
    "Mom": "Julie",
    "phone": 189123456
  }
]
}

```

return all the records from the collection

If there is no cond parameter, the method of `db.collectionSpace.collection.find()` returns all the records from the collection. The following operation select all records in the collection named bar which in the collectionSpace named foo:

```
db.foo.bar.find()
```

return records that match query conditions

- Equality match

The following operation select the records in the collection bar where the age field has the value 23

```
db.foo.bar.find({age: 23})
```

- use [operators](#)

The following operation select the records in the collection bar where the age field is more than(\$gt)20

```
db.foo.bar.find({age: {$gt: 20}})
```

- On array

(1) query an element, The following operation returns a cursor to all records in the bar collection where the array field phone contains the element '10086':

```
db.foo.bar.find({"phone": 10086})
```

(2) query array element which is BSON Object. The following operation return a cursor to all records in the bar collection where family array contains a subdocument element that contains the Dad field equal to 'Kobe' and the phone field equal to '139123456':

```

db.foo.bar.find(
{
  "family": {
    $elemMatch: {
      "Dad": "Kobe",
      "phone": 139123456
    }
  }
})

```

- query on subdocument

The following operation returns a cursor to all documents in the bar collection where the subdocument name is exactly {first:"Tom"}

```

db.foo.bar.find(
{
  "name": {
    "first": "Tom"
  }
}
)

```

It can also be written as :

```
db.foo.bar.find(
  {
    "name.first": "Tom"
  }
)
```

specify the field of the returned records

If specify the sel parameter of find method, then only return the field within sel parameter:

```
db.foo.bar.find(null, {name: ""})
```



**Note:**

If a records doesn't contain the specify field(as:people),Sequoiadb will default return. For example:

```
db.foo.bar.find({}, {name: "", people: ""})
return: {
  "name": {
    "fist": "Tom",
    "second": "David"
  },
  "people": ""
}
```

find() more information

Execute db.foo.bar.find().help() , then will see more methods of find()

- cursor.sort(<sort>)

The method of sort() is used to order by the specified field, syntax: sort({"field1":1|-1,"field2":1|-1,...}), '1' stand for Ascending, '-1' stand for Descending. If the the method of find doesn't specify the content of sel parameter, then sort() will order by the sort parameter, but if sel parameter specify the returned field, then sort() can only order by the field included in the sel parameter. For example:

```
db.foo.bar.find().sort({age:1}) //return all the records in the bar collection and order by the value of age field in ASC
```

```
db.foo.bar.find(null, {name: ""}).sort({age:1}) //this operation actually can't reach the purpose of sorting.
```

- cursor.hint(<hint>)

add indexes to speed up the query speed, assuming there is a index named 'testIndex':

```
db.foo.bar.find().hint({"": "testIndex"})
```

- cursor.limit(<num>)

limit the number of records in the result set:

```
db.foo.bar.find().limit(3) //return the first three records in the result set
```

- cursor.skip(<num>)

the skip() method controls the point of the results set, that is skipping the first 'num' records, begin to return from the 'num+1' records:

```
db.foo.bar.find().skip(5) //skipping the first 5 records in the bar collection
```

- use cursor to control the returned records

```
db.foo.bar.find().current() //return the record that the current cursor point to
```

```
db.foo.bar.find().next() //return the next record that the current cursor point to
```

```
db.foo.bar.find().count() //return the number of records that the current cursor point to
db.foo.bar.find().size() //return the distance from the current cursor to the final cursor
db.foo.bar.find().toArray() //return the result sets with array form
```

## Update

Of the basic database operations(i.e.CRUD),update operation are those that modify the existing records or documents in a collection, using `update()` method to modify documents in Sequoiadb.



Note:

All of the examples in this document use the sequoiadb shell interface.

### update()

the `update()` method is the primary method used to modify documents in Sequoiadb, it has the following syntax :

```
db.collectionspace.collection.update(<rule>, [cond], [hint])
```

The corresponding operation in sql: the method of update is similar with `update...set` statement :

- .the [rule] parameter corresponds to set statement
- .the [cond] parameter corresponds to where statement
- .the [hint] parameter corresponds to the index name in index table

### modify with update operator

If the `update()` method contains only rule parameter expression(such as the `$set` operator expression ), the `update()` method updates the corresponding fields in the document. To update fields in the subdocuments,use dot notation(.).

- update a field in a document

use `$set` to update a value of a field. The following operation queries the bar collection for the document that has an `_id` field equal to 1 and sets the value of field `first` ,in the subdocument name, to 'Mike':

```
db.foo.bar.update ( {$set: {"name.first": "Mike"}}, { _id: 1} )
```



Note:

if the rule parameter contains fields not in the current records,the `update()` method will add the new fields to the records.

- remove a field from a record

use `$unset` to remove the field from a record.The following operation queries the bar collection for all records where contains age field and remove the age field , if a record doesn't contain the age field,skipping.

```
db.foo.bar.update ( {$unset: {age: ""}} )
```

- Update array

if the update operation requires an update of an element in an array field, SequoiaDB use dot notation(.), array are zero-based. The following operation is to update second element in the `arr` array, it's value will increase 5:

```
db.foo.bar.update ( {$inc: {"arr.1": 5}} )
```

- hint parameter

The following operation will traversal all the records through index, and update the value of field name to 'Tom', the name of index is 'textIndex'

```
db.foo.bar.update({$set: {name: "Tom"}}, null, {"": "textIndex"})
```



Note: [more update operator](#)

## Delete

Delete means deleting records in collection. In sequoiadb, [remove\(\)](#) method is used to delete data.



Note:

In this document, all examples use shell interface in sequoiadb.

`remove()`

The method "`remove()`" is used to delete records in collection. The structure of it is:

```
db.collectionspace.collection.remove([cond], [hint])
```

The method "`remove()`" corresponds to the statement "`DELETE`" in SQL:

`.[cond]` corresponds to the statement containing "`where`" in SQL

`. [hint]` corresponds to the name in index table in SQL

Delete Collection Record

- Delete all records in collection

The following command will delete all records in collection "bar":

```
db.foo.bar.remove()
```

- Delete all records that match the condition in collection

The following command will delete all records that contain the value "Tom" in the field "name".

```
db.foo.bar.remove({name: "Tom"})
```

- hint

The following command will rapidly delete all records that contain the value "Tom" in the field "name" after searching them with index. The index name is "textIndex".

```
db.foo.bar.remove({name: "Tom"}, {"": "textIndex"})
```

## SequoiaDB Application Development

---

SequoiaDB allows users to connect it in many ways such as C, C++, Java, PHP, C#, Python. This chapter introduces drivers for different programming language.

### C Driver

This section introduce C driver

[C Driver](#)

[C Environment to Build](#)

[C Development Foundation](#)

[SQL to sequoiadb shell to C](#)

[C API](#)

## C Driver

C client application is mainly to provide database connection, collectionspace, collections, cursor, replica group, node, domain. [More refer to the online C API.](#)

## Handle

The handle of C client-driven is divided into two types. One is used to operate databases, the other is used to operate clusters.

- Database Operation Handle

There are 3 degrees of data in SequoiaDB:

- 1) Database
- 2) Collection space
- 3) Collection

Logically, every collection space in SequoiaDB has no physical upper boundary. Every collection space is stored in a stand-alone file. So the amount of collection space depends on the size of disk and memory.

A collection space contains at most 4096 collections. Every collection can contain more than one record. In a physical system, the size of a collection space is at most 256GB.

Compared with relation database, we can regard record as line in relation database. We can regard collection as table in relation database.

So in client, there are 3 Handles representing connection, collectionspace and collection. The other Handle represents cursor:

sdbConnectionHandle	Database instance	It represents a stand-alone connection
sdbCSHandle	CollectionSpace instance	It represents a stand-alone collection space
sdbCollectionHandle	Collection instance	It represents a stand-alone collection
sdbCursorHandle	Cursor instance	It represents a result set of a query

C client-driven application operate with different handles. For example, reading data will use Cursor Handle, then creating collectionspace uses database connection Handle.



### Note:

1. As for a connection, in its collection space, collections share one socket. So in mutiple-thread system, the system should guarantee that the socket can be used merely by one thread whenever.
2. Generally, it is not recommended to manipulate a connection instance and other instance, which it generates, with more than threads.
3. If each thread merely use its own connection instance and other instances it generates, it is thread-safe.

- Cluster Operation Handle

There are three degrees of cluster operation in SequoiaDB:

- 1) Replica Group
- 2) Data Node
- 3) Domain



### Note:

Replica group contain two tpyes: Catalog ReplicaGroup, Data ReplicaGroup.



ReplicaGroup instances and Data Node instances can be represented by the following three types of Handles.

sdbReplicaGroupHandle	ReplicaGroup Handle	It represents a stand-alone ReplicaGroup
sdbNodeHandle	Data Node Handle	It represents a stand-alone Data Node
sdbDomainHandle	Domain Handle	It represents a stand-alone Domain

Cluster-related operations need to use ReplicaSet Handle and Data Node Handle.

The sdbReplicaGroupHandle instance is to manage replica groups, including starting and stopping replica groups, getting the status, name-information and number-information of nodes in the replica group.

The sdbNodeHandle instance is to manage Data Nodes, including starting and stopping the specified data node, getting the specified data node instance and master-slave data node instance and address-information of data node.

The sdbDomainHandle is used to modify, obtain the domain information.

### Error Information

Each function has its returned value, the definition of the returned value is:

SDB\_OK (value is 0): Execution succeeds.

< 0: Database error, the detailed error description can be found in the include/ossErr.h directory of the C driver development kit.

> 0: System error, please check the related error code information.

### C Environment to Build

#### Get Driver Development Kit

Download SequoiaDBDriver Development kit for the corresponding operating systems from <http://www.sequoiadb.com>.

### Configure development environment

- Linux

1. Extract the Driver Development kit from the downloaded archive.

2. Copy folder named driver from the downloaded archive to development directory (suggest to put on third-party directory), and rename it as "sdbdriver".

3. Add directory named "sdbdriver/include" into compiled directory:

Dynamic Link:

Use lib directory libsdbc.so dynamic library, gcc compiler parameters forms such as:

```
cc testClient.c -o testClientC -I <path>/sdbdriver/include -L <path>/sdbdriver/lib -lsdbc
```

Where: PATH is sdbdriver placed path, When you run the program, you need to specify the path to the LD\_LIBRARY\_PATH contains libsdbc.so dynamic library.

```
export LD_LIBRARY_PATH=<PATH>/sdbdriver/lib
```

- Note:



If there will be an error message when you run the program:

error while loading shared libraries: libsdbc.so: cannot open shared object file: No such file or directory

Said LD\_LIBRARY\_PATH environment variable is not set properly, it is recommended to set to /etc/profile or application startup script, avoiding each new terminal will need to re-open setting.

Static Link:

Use lib directory libstaticsdbc.a static library, gcc compiler parameters forms such as:

```
cc testClient.c -o testClientC -I <path>/sdbdriver/include #L <path>/sdbdriver/lib/
libstaticsdbc.a -lm
```

- Windows

Windows driver development kit is not released yet.

## C Development Foundation

This section introduce the approach to run SequiaDB in C application. First, you should make sure that you have install. Please refer to the [installation page for more about installation information](#).

Edit client code

For the purpose of simpleness, the following sample is not complete code. Please get complete source code in the path "/sequoiadb/client/samples/C".

- Connecting Database

There is a complete client file called "connect.c" to show how to connect to database in a C applicaiton. The file includes a head file called "client.h".

```
#include <stdio.h>
#include "client.h"

// Display Syntax Error
void displaySyntax ( CHAR *pCommand )
{
    printf ( "Syntax: %s<hostname> <servicename> <username> <password>"
            OSS_NEWLINE, pCommand ) ;
}

INT32 main ( INT32 argc, CHAR **argv )
{
    // define a connecion handle; use to connect to database
    sdbConnectionHandle connection = 0 ;
    INT32 rc = SDB_OK ;

    // verify syntax
    if ( 5 != argc )
    {
        displaySyntax ( (CHAR*)argv[0] ) ;
        exit ( 0 ) ;
    }

    // read argument
    CHAR *pHostName = (CHAR*)argv[1] ;
    CHAR *pServiceName = (CHAR*)argv[2] ;
    CHAR *pUsr = (CHAR*)argv[3] ;
    CHAR *pPasswd = (CHAR*)argv[4] ;

    // connect to database
    rc = sdbConnect ( pHostName, pServiceName, pUsr, pPasswd, &connection ) ;
    if ( rc!=SDB_OK )
    {
        printf("Fail to connet to database, rc = %d" OSS_NEWLINE, rc ) ;
        goto error ;
    }
}
```

```

    }

done:
    // disconnect from database
    sdbDisconnect ( connection ) ;
    // release connection
    sdbReleaseConnection ( connection ) ;
    return 0 ;
error:
    goto done ;
}

```

In Linux we can compile and link to dynamically linked librarie file called "libsdbc.so" as follow:

```

$gcc -o connect connect.c -I /path-to/C/include -lsdbc -L/path-to/client/C/lib
$ ./connect localhost 11810 "" ""
connect success!

```



#### Note:

This sample link to the port of 11810 of local database. It log in with null user name and null password. Users need to configurate parameters according to their own situation. But if database has created a user, it can connect to database with it and its password.

- Inserting Data

SequoiaDB stores data in the format of BSON. BSON is a binary object similar to JSON. In order to store data, we should create a BSON object. The following sample insert the record {name:"Tom",age:24} into a collection:

```

// Firstly, a BSON object is created.
INT32 rc = SDB_OK ;
bson obj ;
bson-init( &obj ) ;
bson-append-string( &obj, "name", "tom" ) ;
bson-append-int( &obj, "age", 24 ) ;
rc = bson-finish( &obj ) ;
if ( rc != SDB_OK )
    printf("Error.");
// Secondly, insert it into a collection.
dbInsert ( collection, &obj ) ;

```

- Query

In a query, a cursor handle is used to store the result set of a query. You can get results by manipulating the cursor. This sample uses the sdbNext interface of cursor manipulation to get one record in the result set.

```

// Define a cursor handle.
dbCursorHandle cursor = 0 ;
...
// Search for all records and put the results in the cursor handle.
sdbQuery(collection, NULL, NULL, NULL, NULL, 0, -1, &cursor ) ;
// Show every record in cursor.
bson-init(obj);
while( !( rc=sdbNext( cursor, &obj ) ) )
{
    bson-print( &obj ) ;
    bson-destroy(&obj) ;
    bson-init(&obj);
}
bson-destroy(obj) ;

```

- Index

Here, we create an index in the collection specified by the collection handle collection . It is in ascending order on "name" and descending order on "age".

```

#define INDEX_NAME "index"

```

```

...
// Firstly we create a bson that contains the information of the index.
bson_init( &obj );
bson_append_int( &obj, "name", 1 );
bson_append_int( &obj, "age", -1 );
rc = bson_finish( &obj );
if ( rc != SDB_OK )
printf("Error.");
// Create an index with the BOSN object.
sdbCreateIndex ( collection, &obj, INDEX_NAME, FALSE, FALSE );
bson_destroy ( &obj );

```

- Updating data

Here, we create an index in the collection specified by the collection handle collection This sample doesn't contain any matching condition, so it will update all the records in the collection.

```

// Create a BSON object that contains updating rules
bson_init( &rule );
bson_append_start_object ( &rule, "$set" );
bson_append_int ( &rule, "age", 19 );
bson_append_finish_object ( &rule );
rc = bson_finish ( &rule );
if ( rc != SDB_OK )
printf("Error.");
// Print updating rule.
bson_print( &rule );
// Update records.
sdbUpdate( collection, &rule, NULL, NULL );
bson_destroy(&rule);

```

Here,as no record matching conditions have specified,then this example will update all records in the collection which collection handle specified.

## Cluster Operations

- ReplicaSet Operations

ReplicaSet Operations contains creating ReplicaSets(sdbCreateShard)、 getting ReplicaSet handles(sdbGetShard)、 starting ReplicaSet(sdbStartShard) and stopping ReplicaSet(sdbStopShard). The following shows the ReplicaSet Operations, and real application should include error detecting ,etc.

```

// define a ReplicaSet Handle
sdbShardHandle shard = 0 ;
...
// create a catalog ReplicaSet at first
sdbCreateCataShard ( connection, HOST_NAME, SERVICE_NAME,
                    CATALOG_SET_PATH , NULL );
// sleep 5s to wait starting catalog ReplicaSet and electing master and slave.
sleep( 5 );
// create data ReplicaSet
sdbCreateShard ( connection, GROUP_NAME, &rg );

// create data nodes
sdbCreateNode ( rd, HOST_NAME1, SERVICE_NAME1, DATABASE_PATH1, NULL );
// start ReplicaSet
sdbStartShard( shard );

```

- Data Node Operations

Data Node Operations contains create data node(sdbCreateNode)、 get master data node(sdbGetNodeMaster)、 get slave data node(sdbGetNodeSlave)、 start data node (sdbStartNode)

and stop data node(sdbStopNode), etc. The following shows the Data Node Operations, and real application should include error detecting, etc.

```
// define a data node handle
sdbNodeHandle masternode = 0 ;
sdbNodeHandle slavenode = 0 ;
...
// sleep 10s to wait starting data ReplicaSet and electing master and slave
sleep( 10 ) ;

//get master data node
sdbGetNodeMaster ( shard, &masternode ) ;
//get slave data node
sdbGetNodeSlave ( shard, &slavenode ) ;
```

### SQL to sequoiadb shell to C

The query in Sequoiadb is in the format of json(bson). The following table shows examples comparing sql statements, sequoiadb shell statements and sequoiadb C driver application statements.

SQL	sequoiadb shell	java Driver
insert into students(a,b) values(1,-1)	db.foo.bar.insert({a:1,b:-1})	const char *r="{a:1,b:-1}"; // JsonToBso transforms a json string into a bson object jsonToBson ( &obj, r ); // Collection is the handle of collection bar sdbInsert (collection, &obj );
select a,b from students	db.foo.bar.find(null,{a:"",b:""})	const char *r="{a:W\"W\",b:W\"W\"}"; // JsonToBso transforms a json string into bson object jsonToBson ( & select, r ); // Collection is the handle of collection bar. Cursor is the handle that returns query results. sdbQuery ( collection, NULL, &select, NULL, NULL, 0, -1, cursor );
select * from students	db.foo.bar.find()	// Collection is the handle of collection bar. Cursor is the handle that returns query results. sdbQuery ( collection, NULL, NULL, NULL, NULL, 0, -1, cursor );
select * from students where age=20	db.foo.bar.find({age:20})	const char *r="{age:20}"; // JsonToBso transforms a json string into a bson object. jsonToBson ( &condition, r ); // Collection is the handle of collection bar. Cursor is the handle that returns query results. sdbQuery ( collection, & condition , NULL, NULL, NULL, 0, -1, cursor );
select * from students where age=20 order by name	db.foo.bar.find({age:20}).sort({name:1})	const char *r1="{age:20}"; const char *r2="{name:1}"; // JsonToBso transforms a json string into a bson object jsonToBson ( & condition, r1 ); jsonToBson ( &orderBy, r2 ); // Collection is the handle of collection bar. Cursor is the handle that returns query results. sdbQuery ( collection, & condition , NULL, & orderBy , NULL, 0, -1, cursor );
select * from students where age>20 and age<30	db.foo.bar.find({age:{\$gt:20,\$lt:30}})	const char *r="{age:{\$gt:20,\$lt:30}}"; // JsonToBso transforms a json string into a bson object jsonToBson ( &condition, r );

SQL	sequoiadb shell	java Driver
		// Collection is the handle of collection bar. Cursor is the handle that returns query results. sdbQuery ( collection, & condition , NULL, NULL, NULL, 0, -1, cursor ) ;
create index testIndex on students(name)	db.foo.bar.createIndex("testIndex", {name:1},false)	const char *r="{name:1}"; // jsonToBson transforms a json string into a bson object jsonToBson ( &obj, r ) ; // Collection is the handle of collection bar sdbCreateIndex ( collection, &obj, "testIndex", FALSE, FALSE )
select * from students limit 20 skip 10	db.foo.bar.find().limit(20).skip(10)	// Collection is the handle of collection bar. Cursor is the handle that returns query results. sdbQuery ( collection, NULL, NULL, NULL,NULL, 10, 20, cursor ) ;
select count(*) from students where age>20	db.foo.bar.find({age:{\$gt:20}}).count()	const char *r="{age:{\$gt:20}}"; // jsonToBson transforms a json string into a bson object jsonToBson ( &condition, r ) ; // Collection is the handle of collection bar. Count is the total amount of results. sdbGetCount ( collection, & condition , &count );
update students set a=a+2 where b=-1	db.foo.bar.update({\$set:{a:2}},{b:-1})	const char *r1="{ \$set:{a:2}}"; const char *r2="{b:-1}"; // jsonToBson transforms a json string into a bson object jsonToBson ( &rule, r1 ) ; jsonToBson ( &condition, r2 ) ; // Collection is the handle of collection bar. sdbUpdate (collection, &rule, &condition, NULL ) ;
delete from students where a=1	db.foo.bar.remove({a:1})	const char *r="{a:1}"; // jsonToBson transforms a json string into a bson object jsonToBson ( &condition, r ) ; // Collection is the handle of collection bar. sdbDelete ( collection, & condition , NULL ) ;

## C API

This part is related to C API document.

## C API

### History Updates:

#### Version1.8

#### New APIs in C:

sdbConnect1, multiple connection address can be provided, one of the random valid addresses will be used to connect to database  
sdbCreateCollectionSpaceV2, a BSON parameter is provided to make the creating of a collection space more flexible  
sdbAlterCollection, alters the attributes of a collection  
sdbCreateDomain, creates a domain  
sdbDropDomain, drops a domain  
sdbGetDomain, gets a domain  
sdbListDomains, enumerates and lists the domains  
sdbReleaseDomain, releases the domain handle  
sdbAlterDomain, alters the attributes of a domain

Version1.6

1. Use `sdbNodeHandle` to replace the original `sdbReplicaNodeHandle`. `sdbReplicaNodeHandle` will be abandoned in the version2.x.
2. Use the concept of "node" to replace the original "replica node". And "replica node" related API interface will remain until version2.x will be abandoned.

For details, please see the relevant API.

## C++ Driver

This section introduce C++ driver.

[C++ Driver](#)

[C++ Environment to Build](#)

[C++ Development Foundatio](#)

[SQL to sequoiadb shell to C++](#)

[C++ API](#)

### C++ Driver

C++ client-driven provide interfaces of database operations and cluster operations. Mainly include the following 7 levels: database operation, collection spaces, collections, cursor, replica group, node, domain.

[For more please refer to online C++ API.](#)

### C++ Class Instances

C++ Client-Driven has two class instances. One is used to operate database, the other is used to operate cluster.

- Database operation instance

There are 3 degrees of data in SequoiaDB:

- 1) Database
- 2) Collection space
- 3) Collection

So in client, there are 3 Handles representing connection, collectionspace and collection. The other Handle represents cursor:

Sdb	Database class	it represents a stand-alone database connection
sdbCollectionSpace	CollectionSpace class	it represents a stand-alone collection space
sdbCollection	Collection class	it represents a stand-alone collection
sdbCursor	Cursor class	it represents a result set of a query

C++ client-driven application operate with different instances. For example, reading data will use Cursor instance, then creating collectionspace uses database connection instance.

- Cluster Operation Instance

There are three degrees of cluster operation in SequoiaDB:

- 1) ReplicaGroups
- 2) Data Node
- 3) Domain

**Note:**

ReplicaGroups contain three types: Coord ReplicaGroups, Catalog ReplicaGroup, Data ReplicaGroup.

ReplicaGroup instances, Data Node instances and domain instances can be represented by the following three types of classes.

sdbReplicaGroup	Replica group class	It represents a stand-alone ReplicaGroup
sdbNode	data node class	It represents a stand-alone Data Node
sdbDomain	domain class	It represents a stand-alone Domain

Cluster-related operations need to use ReplicaGroup instance and Data Node instance.

The sdbReplicaGroup instance is to manage replica groups, including starting and stopping replicagroups, getting the status, name-information and number-information of nodes in the replicagroup.

The sdbNode instance is to manage Data Nodes, including starting and stopping the specified data node, getting the specified data node instance and master-slave data node instance and address-information of data node.

The sdbDomain is used in the management domain, The modification of the domain, obtain the domain information such as operation.

**Error Information**

Each function has its returned value, the definition of the returned value is:

SDB\_OK (value is 0): Execution succeeds.

< 0: Database error, the detailed error description can be found in the include/ossErr.h directory of the C++ driver development kit.

> 0: System error, please check the related error code information.

**C++ Environment to Build****Get Driver Development Kit**

Download SequoiaDB Driver Development kit for the corresponding operating systems from <http://www.sequoiadb.com>.

**Configure development environment**

- Linux

1. Extract the Driver Development kit from the downloaded archive.

2. Copy folder named driver from the downloaded archive to development directory (suggest to put on third-party directory), and rename it as "sdbdriver".

3. Add directory named "sdbdriver/include" into compiled directory.

Dynamic Link:

Use lib directory libsdbcpp.so dynamic library, gcc compiler parameters forms such as:

```
g++ main.cpp -o test -I <PATH>/sdbdriver/include -L <PATH>/sdbdriver/lib -lsdbcpp
```

Where: PATH is sdbdriver placed path, When you run the program, you need to specify the path to the LD\_LIBRARY\_PATH contains libsdbcpp.so dynamic library.

```
export LD_LIBRARY_PATH=<PATH>/sdbdriver/lib
```



-  Note:

If there will be an error message when you run the program:

error while loading shared libraries: libsdbcpp.so: cannot open shared object file: No such file or directory

Said LD\_LIBRARY\_PATH environment variable is not set properly, it is recommended to set to /etc/profile or application startup script, avoiding each new terminal will need to re-open setting.

Static Link:

Use lib directory libstaticsdbc.a static library, gcc compiler parameters forms such as:

```
g++ main.c -o test -I <path>/sdbdriver/include #L <path>/sdbdriver/lib/libstaticsdbc.a #
lm -lpthread
```

- Windows

Windows driver development kit is not released yet.

## C++ Development Foundation

This section introduce how to use C++ client-driven interfaces to run C++ application. for simple, the following examples are not full code, only play an exemplary role. You can get the full code in the directory "/sequoiadb/client/samples/CPP", for more, please refer to [online C++ API](#)

## Database operations

- Connecting Database

"connect.cpp" shows how to connect database. The file must contain the head file "client.hpp" and use namespace "sdbclient".

```
#include <stdio.h>
#include <iostream>
#include "client.hpp"

using namespace std;
using namespace sdbclient;

// Display Syntax Error
void displaySyntax ( CHAR *pCommand ) ;

INT32 main ( INT32 argc, CHAR **argv )
{
    // define a sdb object
    // use to connect to database
    sdb connection;
    INT32 rc = SDB_OK;

    // verify syntax
    if ( 5 != argc )
    {
        displaySyntax ( (CHAR*)argv[0] );
        exit ( 0 );
    }

    // read argument
    CHAR *pHostName    = (CHAR*)argv[1];
    UINT16 Port         = atoi (argv[2]);
    CHAR *pUsr         = (CHAR*)argv[3];
    CHAR *pPasswd       = (CHAR*)argv[4];
```

```

// connect to database
rc = connection.connect ( pHostName, Port, pUsr, pPasswd ) ;
if ( rc!=SDB_OK )
{
    cout<<"Fail to connet to database, rc = "<<rc<<endl ;
    goto error ;
}
else
    cout<<"connect success!"<<endl ;
done:
// disconnect from database
connection.disconnect () ;
return 0 ;
error:
goto done ;
}

// Display Syntax Error
void displaySyntax ( CHAR *pCommand )
{
    cout<<"Syntax: "<<pCommand<<" <host><name><servicename>\
<username><password>"<<endl ;
}

```

In Linux, you can compile and link the dynamic library "libsdbcpp.so" like this:

```

$ gcc -o connect connect.cpp -I <PATH>/sequoiadb/client/CPP/include -lsdbcpp -L <PATH>/
sequoiadb/client/CPP/lib
Execution results:
$ ./connect localhost 11810 "" ""
connect success!

```

In Windows, compiling and linking the dynamic library "sdbcpp.lib" and "sdbcpp.dll" like this:

```

> cl /Fo connect.obj /c connect.cpp /I <PATH>\sequoiadb\client\CPP\include /wd4047
> link /OUT:connect.exe /LIBPATH:<PATH>\sequoiadb\client\CPP\lib sdbcpp.lib connect.obj
> copy <PATH>\sequoiadb\client\CPP\lib\sdbcpp.dll .
Execution results:
> connect.exe localhost 11810 "" ""
> connect success!

```



#### Note:

This sample connects to the port "11810" of local database, username and password are null, users can configure params according to needs. For example, when the user have created username and password, you must use the correct username and password to connect the database.

- Inserting Data

```

// Firstly , a BSON object is created
bson obj;
bson-init( &obj ) ;
bson-append-string( &obj, "name", "tom" ) ;
bson-append-int( &obj, "age", 24 ) ;
bson-finish( &obj ) ;
// Then,insert it into a collection
collection.insert ( &obj ) ;
bson-destroy( &obj ) ;

```

obj is inputed param and inserted data.

- Query

```

// Define a cursor object
sdbCursor cursor ;
...
// Search for all records and put the results in the cursor object

```

```
collection.query ( cursor, NULL, NULL, NULL, NULL, 0, -1 ) ;
// Show every record in cursor
bson_init(&obj);
while( !( rc=cursor.next( obj ) ) )
{
    bson_print( &obj ) ;
    bson_destroy(&obj) ;
    bson_init(&obj);
}
bson_destroy(&obj) ;
```

In a query, a cursor object is used to store the result set of a query . You can get results by manipulating the cursor. This sample uses the next interface of cursor manipulation to get one record in the result set. This sample does not set querying conditions, filtering conditions and ordering, only use default index.

- Index

```
#define INDEX_NAME "index"
...
// Firstly we create a bson that contains the information of the index.
bson_init( &obj ) ;
bson_append_int( &obj, "name", 1 ) ;
bson_append_int( &obj, "age", -1 ) ;
bson_finish( &obj ) ;
// Create an index with the BOSN object.
collection.createIndex ( &obj, INDEX_NAME, FALSE, FALSE ) ;
bson_destroy ( &obj ) ;
```

Here, we create an index in the collection specified by the collection object collection . It is in ascending order on "name" and descending order on "age"

- Update

```
// Create a BSON object that contains updating rules
bson_init( &rule ) ;
bson_append_start_object ( &rule, "$set" ) ;
bson_append_int ( &rule, "age", 19 ) ;
bson_append_finish_object ( &rule ) ;
bson_finish ( &rule ) ;
// Print updating rule.
bson_print( &rule ) ;
// Update records.
collection.update( &rule, NULL, NULL ) ;
bson_destroy(&rule);
```

Here, as no record matching conditions have specified, then this example will update all records in the collection which collection handle specified.

## Cluster Operations

- ReplicaSet Operations

ReplicaSet Operations contains creating ReplicaSets(sdbCreateShard), getting ReplicaSet handles(sdbGetShard), starting ReplicaSet(sdbStartShard) and stopping ReplicaSet(sdbStopShard). The following shows the ReplicaSet Operations, and real application should include error detecting, etc.

```
// define a ReplicaSet Instance
sdbShard shard ;
// define a null map object, it means that no more configuration information when create data nodes
map<string, string> config ;
...
// create a catalog ReplicaSet at first
```

```

connection.createCataShard ( HOST_NAME, SERVICE_NAME,
                             CATALOG_SHARD_PATH , NULL ) ;
// sleep 5s to wait starting catalog ReplicaSet and electing master and slave.
sleep( 5 ) ;
// create data ReplicaSet
connection.createShard ( REPLICAS_SHARD_NAME, shard ) ;

// create data nodes
shard.createNode ( HOST_NAME1, SERVICE_NAME1, DATABASE_PATH1, config ) ;
...
// start ReplicaSet
shard.start () ;

```

- Data Node Operations

Data Node Operations contains create data node(sdbShardcreateNode)、get master data node(sdbShardgetMaster)、get slave data node(sdbShardGetSlave)、start data node (sdbNodeStop) and stop data node(sdbNodeStop),etc. The following shows the Data Node Operations, and real application should include error detecting, etc.

```

// define a data node instance
sdbNode masternode ;
sdbNode slavenode ;
...
// sleep 15s to wait starting data ReplicaSet and electing master and slave
sleep( 15 ) ;

// get master data node
shard.getMaster( masternode ) ;
//get slave data node
shard.getSlave( slavenode ) ;

```

### SQL to sequoiadb shell to C++

The query in Sequoiadb is in the format of json(bson). The following table shows examples comparing sql statements, sequoiadb shell statements and sequoiadb C++ driver application statements.

SQL	sequoiadb shell	C++ Driver
insert into students(a,b) values(1,-1)	db.foo.bar.insert({a:1,b:-1})	sdbCollection collection ; const char *r="{a:1,b:-1}"; jsonToBson ( &obj, r ) ; collection.insert( &obj ) ;
select a,b from students	db.foo.bar.find(null,{a:"",b:""})	sdbCollection collection ; sdbCursor cursor ; const char *r="{a:W\"W\",b:W\"W\"}"; jsonToBson ( &select, r ) ; collection .query( cursor, NULL, &select, NULL, NULL, 0, -1 ) ;
select * from students	db.foo.bar.find()	sdbCollection collection ; sdbCursor cursor ; collection .query (cursor, NULL, NULL, NULL, NULL, 0, -1 ) ;
select * from students where age=20	db.foo.bar.find({age:20})	sdbCollection collection ; sdbCursor cursor ; const char *r="{age:20}"; jsonToBson ( &condition, r ) ; collection .query (cursor, &condition , NULL, NULL, NULL, 0, -1 ) ;
select * from students where age=20 order by name	db.foo.bar.find({age:20}).sort({name:1})	sdbCollection collection ; sdbCursor cursor ; const char *r1="{age:20}"; const char *r2="{name:1}";

SQL	sequoiadb shell	C++ Driver
		<pre> jsonToBson ( &amp; condition, r1 ); jsonToBson ( &amp;orderBy, r2 ); collection .query (cursor, &amp; condition , NULL, &amp; orderBy , NULL, 0, -1 ); </pre>
select * from students where age>20 and age<30	db.foo.bar.find({age:{\$gt:20,\$lt:30}})	<pre> sdbCollection collection ; sdbCursor cursor ; const char *r ="{age:{\$gt:20,\$lt:30}}"; jsonToBson ( &amp;condition, r ); collection .query (cursor, &amp; condition , NULL, NULL, NULL, 0, -1 ); </pre>
create index testIndex on students(name)	db.foo.bar.createIndex("testIndex", {name:1},false)	<pre> sdbCollection collection ; const char *r ="{name:1}"; jsonToBson ( &amp;obj, r ); collection.createIndex ( &amp;obj, "testIndex", FALSE, FALSE ) </pre>
select * from students limit 20 skip 10	db.foo.bar.find().limit(20).skip(10)	<pre> sdbCollection collection ; sdbCursor cursor ; collection .query (cursor, NULL, NULL, NULL, NULL, 10, 20 ); </pre>
select count(*) from students where age>20	db.foo.bar.find({age:{\$gt:20}}).count()	<pre> sdbCollection collection ; SINT64 count = 0 ; const char *r ="{age:{\$gt:20}}"; jsonToBson ( &amp;condition, r ); collection.getCount ( collection, &amp; condition , &amp;count ); </pre>
update students set a=a+2 where b=-1	db.foo.bar.update({\$set:{a:2}},{b:-1})	<pre> sdbCollection collection ; const char *r1 ="{\$set:{a:2}}"; const char *r2 ="{b:-1}"; jsonToBson ( &amp;rule, r1 ); jsonToBson ( &amp;condition, r2 ); collection.update ( &amp;rule, &amp;condition, NULL ); </pre>
delete from students where a=1	db.foo.bar.remove({a:1})	<pre> sdbCollection collection ; const char *r ="{a:1}"; jsonToBson ( &amp;condition, r ); collection.del ( &amp; condition , NULL ); </pre>

## C++ API

This part is related to the C++ API document.

## C++ API

### History Updates:

#### Version1.8

##### 1. New APIs in sdb Class:

```

connect, multiple connection address can be provided, one of the random valid addresses will
be used to connect to database
createCollectionSpace, a BSON parameter is provided to make the creating of a collection space
more flexible
backupOffline, more options are supported by backupOffline()
createDomain, creates a domain
getDomain, gets a domain
dropDomain, drops a domain
listDomain, enumerates and lists the domains

```

## 2. New APIs in sdbCollection Class:

`alterCollection`, alters the attributes of a domain

## 3. A Domain Class is added to support the domain-related operations

Version1.6

1. Add a class Node to replace the original class ReplicaNode. Class ReplicaNode and methods associated with it will be abandoned in the version 2.x.

For details, please see the relevant API.

## Java Driver

This section introduce Java driver.

[Java Driver](#)

[Java Environment to Build](#)

[Java Development Foundation](#)

[Java CSON Use](#)

[Java datasource Introduction](#)

[SQL to sequoiadb shell to java](#)

[Java API](#)

Java Driver

### Outline

SequoiaDB Java driver provides an interface database operations, and cluster operations. Includes the following seven levels of operation: database, collection space, collection, cursor, replica group, node, domain.

There are two driving Java class instances. A database operation, another operation for the cluster.

- Class sample

There are 3 degree of data in SequoiaDB:

1) Database

2) Collection space

3) Collection

Logically, every collection space in SequoiaDB has no physical upper boundary. Every collection space is stored in a stand-alone file. So the amount of collection space depends on the size of disk and memory.

A collection space contains at most 4096 collections. Every collection can contain more than one record. In a physical system, the size of a collection space is at most 256GB.

Compared with relation database, we can regard record as line in relation database. We can regard collection as table in relation database.

So in client, there are 3 classes representing 3 kinds of objects. The other class represents cursor:

Sequoiadb	Database instance	It represents a stand-alone connection
CollectionSpace	Collection space instance	It represents a stand-alone collection space
DBCcollection	Collection instance	It represents a stand-alone collection
DBCcursor	Cursor Instance	It represents a result set of a query



Note:

SequoiaDB only builds one Socket connection, and it does not lock the network operation internally. If more threads are needed to connect the database, each thread must create a new Sequoiadb object by itself respectively, and also the CollectionSpace/DBColltion/DBCursor objects belong to each Sequoiadb.

- Cluster Operation

SequoiaDB database mainly has three levels of cluster operations:

- 1) Replica Group
- 2) Data Node
- 3) Domain



Note: There are two types of replica group: Catalog Replica Group and Data Replica Group.

Replica group instance and data node instance can be represented by the instance of the following three classes.

ReplicaGroup	Replica group class	Each instance stands for a single replica group
Node	Data node class	Each instance stands for a single data node
Domain	Domain class	Each instance stands for a domain which

Replica group and data node instances are needed for the cluster-related operations.

A ReplicaGroup instance is used to manipulate the replica groups. Its operations include start/stop the replica groups, get the status, name, and amount of the nodes in the replica groups.

A Node instance is used to manipulate the node. Its operations include start/stop the node, get the specified node instances, obtain the master-slave node example, get data node address information.

A sdbDomain instance is used to manipulate the domain. Its operations include modify the domains, and get the information of the domains.

## Error Information

- When execution exceptions occur, most APIs will throw `com.sequoiadb.exception.BaseExceptio` or `java.lang.Exception`, correspond to the exception info returned from the database engine or returned from local client respectively.
- `BaseException`'s exception detailed info can be gotten by the class's `getErrorType()`, `getErrorCode()` and `getMessage()` methods.

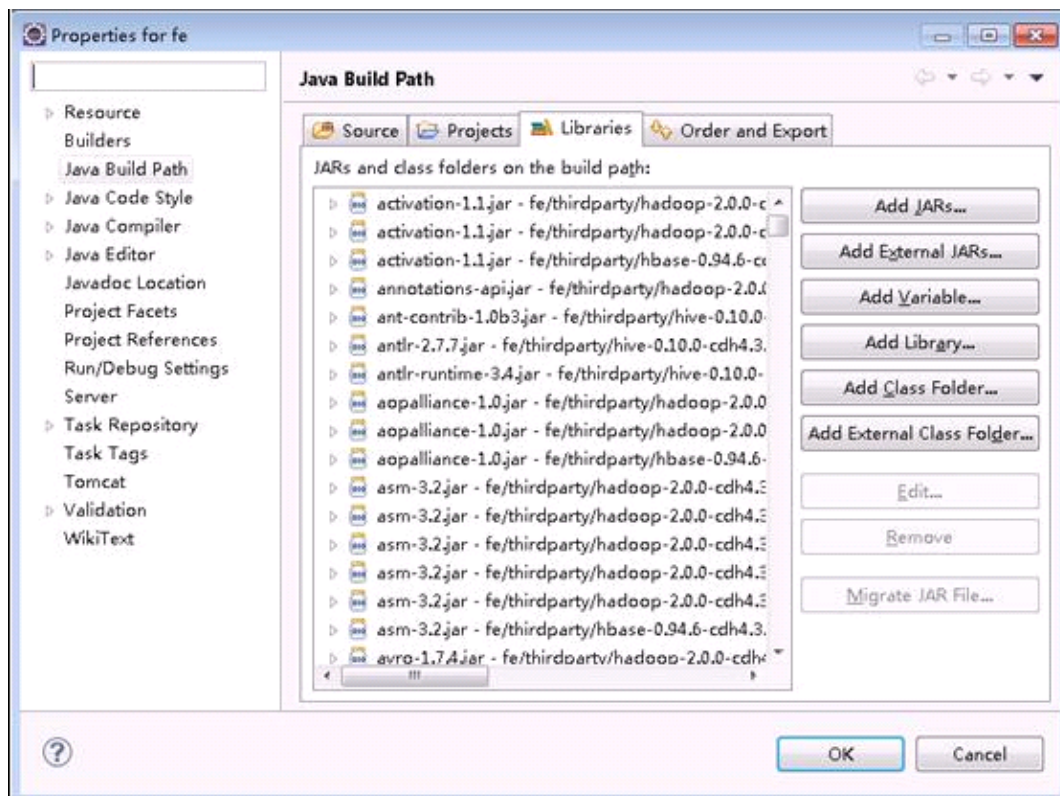
## Java Environment to Build

### Getting the Driver Development Kit

From <http://www.sequoiadb.com> download the corresponding operating system version SequoiaDB Driver Development Kit, extract the driver development kit, from `driver/java/` directory access `sequoiadb.jar` file.

### Configuring Eclipse development environment

- The SequoiaDB driver development package `sequoiadb.jar` files are copied to the project file directory (recommended be placed on all other dependent libraries directory, such as `lib` directory).
- In Eclipse interface, create/open development engineering.
- The left side of the main window in the Eclipse "Package Explore" window, select development projects, and click the right mouse button.
- In the menu, select "properties" menu item.
- In the pop-up "property for project ..." window, select "Java Build Path" -> "Libraries", as shown below:



6. Click on "Add JARs.." button, select Add sequoiadb.jar to the project.
7. Click "OK" to complete environment configuration.

[Please refer here for more](#)

### Java Development Foundation

This section introduce how to use Java client-driven interfaces to run Java application. for simple, the following examples are not full code, only play an exemplary role. You can get the full code in the directory `"/sequoiadb/client/samples/java"`, for more, please refer to [online Java API](#)

### Database Operations

- Database Connection: The following codes connect to a database and list all the collection information:

```
import com.sequoiadb.base.DBCursor;
import com.sequoiadb.base.Sequoiadb;
import com.sequoiadb.exception.BaseException;

public class Sample {
    public static void main(String[] args) {

        String connString = "192.168.1.2:11810";
        try {
            // build a Sequoiadb database connection
            Sequoiadb sdb = new Sequoiadb(connString, "", "");
            // get and print all the Collection information
            DBCursor cursor = sdb.listCollections();
            while(cursor.hasNext()) {
                System.out.println(cursor.getCurrent());
            }
        } catch (BaseException e) {
            System.out.println("Sequoiadb driver error, error description: " + e.getErrorType());
        }
    }
}
```



```

}
}
}

```

**Note:**

1) In the above sample, the port 11810 the local database connects to is the service port of a coord node; it uses null for user name and password. The users need to configure the parameters according to their own situations.

2) SequoiaDB class is a non thread-safe class; each thread must build its own SequoiaDB object, and cannot be passed to multithreads at the same time.

- **Insert Data**

```

String connString = "192.168.1.2:11810";
try {

    Sequoiadb sdb = new Sequoiadb(connString, "", "");

    CollectionSpace db = sdb.createCollectionSpace("space");
    DBCollection cl = db.createCollection("collection");

    // Create a bson object as an insertor
    BSONObject obj = new BasicBSONObject();
    obj.put("name", "tom");
    obj.put("age", 24);

    cl.insert(obj);

} catch (BaseException e) {
    System.out.println("Sequoiadb driver error, error description: " + e.getErrorType());
}

```



**Note:** In the above sample, the port 11810 the local database connects to is the service port of a coord node; it uses null for user name and password. The users need to configure the parameters according to their own situations.

- **Query Data**

```

// define a cursor object
DBCursor cursor;

BSONObject queryCondition = new BasicBSONObject();
queryCondition = (BSONObject) JSON.parse("({age: {$ne: 20}})");
// Query all the records, and put the result set into the cursor object
cursor = cl.query(queryCondition, null, null, null);
// print all the records in the cursor
while (cursor.hasNext()) {
    BSONObject record = cursor.getNext();
    String name = (String) record.get("name");
    System.out.println("name=" + name);
}

```

**Note:**

1) The above sample sets some simple query conditions. And some other parameters include select condition, sorting order, etc.

2) The cursor object buffers part of the data into the local process's memory, it will get other parts of the data from the server through the network and buffers the data locally again.

## Cluster Operations

- **Create Replica Group**

```
String connString = "192.168.1.2:11810";
```

```
try {
    Sequoiadb sdb = new Sequoiadb(connString, "", "");
    ReplicaGroup rg = sdb.createRG("group1");
    rg.createNode("dbserver-1", 11820, "/opt/sequoiadb/database/data/11820", null);
    rg.start();
} catch (BaseException e) {
    System.out.println("Sequoiadb driver error, error description" + e.getErrorType());
}
```

**Note:**

1) rg.createNode() method's first parameter is the host name (IP address is not supported currently) of the server of newly added node. The third parameter is the path which the data files store in, SequoiaDB will automatically create that directory, and please make sure that the SequoiaDB administrator account (sdbadmin in default) has the write permission.

2) rg.start() method is used to start all the nodes in a replica group. This function usually needs 10s to process. This method does not guarantee that the master node selection has finished, so in order to guarantee the replica group work properly, before using the new replica group, please wait another 30s after the start() method finished.

- Add a New Node in the Replica Group

```
String connString = "192.168.1.2:11810";
try {
    Sequoiadb sdb = new Sequoiadb(connString, "", "");

    ReplicaGroup rg = sdb.getReplicaGroup("group1");
    Node node = rg.createNode("dbserver-1", 11830, "/var/sequoiadb/database/data/11830",
null);
    node.start();
} catch (BaseException e) {
    System.out.println("Sequoiadb driver error, error description" + e.getErrorType());
}
```

**Note:**

rg.createNode() method's first parameter is the host name (IP address is not supported currently) of the server of newly added node.

## Java BSON Usage

### Java BSON data type

SequoiaDB supports various BSON data type. For more details, please check [Database Concept-Database-Document](#) section.

### Java BSON Structured Data Type

- Integer/Float

Java BSON structured integer/float type:

```
// {a: 123, b: 3.14}
BSONObject obj = new BasicBSONObject();
obj.put("a", 123);
obj.put("b", 3.14);
```

- String

Java BSON structured string type:

```
// {a: "hi"}
BSONObject obj = new BasicBSONObject();
obj.put("a", "hi");
```

- null

Java BSON structured null type:

```
// {a:null}
BSONObject obj = new BasicBSONObject();
obj.put("a", null);
```

- object

Java BSON structured nested object type:

```
// {b: {a:1}}
BSONObject subObj = new BasicBSONObject();
subObj.put("a", 1);
BSONObject obj = new BasicBSONObject();
obj.put("b", subObj);
```

- array

Java BSON uses org.bson.types.BasicBSONList to structure the array type:

```
// {a: [0,1,2]}
BSONObject obj = new BasicBSONObject();
BSONObject arr = new BasicBSONList();
arr.put("0", 0);
arr.put("1", 1);
arr.put("2", 2);
obj.put("a", arr);
```

- Boolean

Java BSON structured boolean type:

```
// {a:true, b:false}
BSONObject obj = new BasicBSONObject();
obj.put("a", true);
obj.put("b", false);
```

- objectID

Java BSON uses org.bson.types.ObjectId to generate the value of each "\_id" field. Java BSON's ObjectId is 12 bits long which is a little different from the ObjectId introduced in [Database Concept-Database-Document-ObjectId](#) section, currently, the 12 bits of a Java ObjectId are composed of three parts: 4 bits for a timestamp accurate to milliseconds, 4 bits for the identification of the system (physical machine).

```
BSONObject obj = new BasicBSONObject();
ObjectId id1 = new ObjectId();
ObjectId id2 = new ObjectId("53bb5667c5d061d6f579d0bb");
obj.put("_id", id1);
```

- regex

Java Bson uses java.util.regex. Pattern to structure the regex data type.

```
BSONObject matcher = new BasicBSONObject();
Pattern obj = Pattern.compile("^2001", Pattern.CASE_INSENSITIVE);
matcher.put("serial_num", obj);
BSONObject modifier = new BasicBSONObject("$set", new BasicBSONObject("count", 1000));
cl.update(matcher, modifier, null);
```

The above regex structures a matching condition; it modifies the value of the "count" field of all the records with a serial number that starts with 2001 into 1000.



Note:

The above uses Pattern to structure the bson matcher, but if using matcher.toString(), the content will be:

```
{ "serial_num" : { "$options" : "i" , "$regex" : "^2001"}}
```

The same content can also be obtained by using the following bson structuring method:

```
BSONObject matcher2 = new BasicBSONObject();
```

```

BSONObject obj2 = new BasicBSONObject();
obj2.put("$regex", "^2001");
obj2.put("$options", "i");
matcher2.put("serial_num", obj2);

```

But matcher2 structured in the latter way is a common nested type instead of a regex.

- **Date**

Java BSON uses java.util.Date to structure the date type:

```

BSONObject obj = new BasicBSONObject();
Date now = new Date();
obj.put("date", now);

```

- **Binary**

Java BSON uses java.util.type.Binary to structure the binary type:

```

BSONObject obj = new BasicBSONObject();
String str = "hello world";
byte[] arr = str.getBytes();
Binary bindata = new Binary(arr);
obj.put("bindata", bindata);

```

- **TimeStamp**

Java BSON uses org.bson.types.BSONTimestamp to structure the time stamp type:

```

int sec = 1404189030 ; // 2014-07-01 12:30:30
BSONObject obj = new BasicBSONObject();
BSONTimestamp ts = new BSONTimestamp(sec, 0);
obj.put("timestamp", ts);

```

## Java datasource Introduction

The datasource in the Java Driver provides users a quick access to obtain valid connetions.

### The usage of Connection pool

Use the getConnection() method in SequoiadbDatasource Class to get a connection from the connection pool, use the close() method to return the connections back to the pool. When the connections in the pool reach the upper limit, next request will wait for a little while, if there are no idle connections available in the given time, an exception will be thrown out. ConfigOptions Class can set different parameters of the connections. SequoiadbOption Class can set different parameters of the connection pool.

For more details, please check the introduction of related APIs.

### Examples:

```

SequoiadbDatasource ds = null;
Sequoiadb db = null;
ArrayList<String> urls = new ArrayList<String>();
ConfigOptions nwOpt = new ConfigOptions(); // define connection options
SequoiadbOption dsOpt = new SequoiadbOption(); // define connection pool options

urls.add("ubuntu-dev1:11810");
urls.add("ubuntu-dev2:11810");
urls.add("ubuntu-dev3:11810");

nwOpt.setConnectTimeout(500); // set connection timeout(ms)
nwOpt.setMaxAutoConnectRetryTime(0); // set maximum auto-connection-retry times

// The following are the default setups of SequoiadbOption.
dsOpt.setMaxConnectionNum(500); // set the maximum number of connections
// in the pool
dsOpt.setInitConnectionNum(10); // Initialize the connection pool, set the
// initial connection number

```

```

dsOpt.setDeltaIncCount(10) ; // when no connections are available in
    the pool, increases the maximum number of connections
dsOpt.setMaxIdleNum(10); // set the maximum idle connections when
    cleaning the extra idle connections periodically
dsOpt.setTimeout(5 * 1000); // set the time of waiting connections
    when the number of used connections reaches the upper limit(500)
dsOpt.setAbandonTime(10 * 60 * 1000); // set the abandon time. A connection will
    be abandoned when the idle time of a connection is longer than the abandon time.
dsOpt.setRecheckCyclePeriod(1 * 60 * 1000); // set the cycle period of cleaning the
    extra idle connections
dsOpt.setRecaptureConnPeriod(10 * 60 * 1000); // set the cycle period of checking and
    capturing the abnormal connections

ds = new SequoiadbDatasource(urls, "", "", nwOpt, dsOpt); // create a connection pool
db = ds.getConnection(); // get connections from the pool
// do something else // use the connections to finish
    certain operations
ds.close(db);

```

## SQL to sequoiadb shell to Java

The query in Sequoiadb is in the format of json(bson). The following table shows examples comparing sql statements, sequoiadb shell statements and sequoiadb java driver application statements.

SQL	sequoiadb shell	java Driver
insert into students(a,b) values(1,-1)	db.foo.bar.insert({a:1,b:-1})	bar.insert("{\"a\":1,\"b\":-1}")
select a,b from students	db.foo.bar.find(null,{a:"",b:""})	bar.query("", "{a:'',b:''}", "", "")
select * from students	db.foo.bar.find()	bar.query()
select * from students where age=20	db.foo.bar.find({age:20})	bar.query("{\"age\":20}", "", "", "")
select * from students where age=20 order by name	db.foo.bar.find({age:20}).sort({name:1})	bar.query("{\"age\":20}", "", "{\"name\":1}", "")
select * from students where age>20 and age<30	db.foo.bar.find({age:{>:20,<:30}})	bar.query("{\"age\":{\">\":20,\"<\":30}}", "", "", "")
create index testIndex on students(name)	db.foo.bar.createIndex("testIndex", {name:1},false)	bar.createIndex("testIndex", "{name:1}", false, false)
select * from students limit 20 skip 10	db.foo.bar.find().limit(20).skip(10)	bar.query("", "", "", "", 10, 20)
select count(*) from students where age>20	db.foo.bar.find({age:{>:20}}).count()	bar.getCount("{\"age\":{\">\":20}}")
update students set a=a+2 where b=-1	db.foo.bar.update({\$set:{a:2}},{b:-1})	bar.update("{\"b\":-1}", "{\$inc\":{\"a\":2}}", "")
delete from students where a=1	db.foo.bar.remove({a:1})	bar.delete("{\"a\":1}")

## Java API

This part is related to Java API document.

### [Java API](#)

## History Updates:

### Version1.8

#### 1. New APIs in Sequoiadb Class:

```

isValid, checks whether the current connection is valid or not
createCollectionSpace, a BSONObject parameter is provided to make the creating of a collection
    space more flexible
backupOffline, more options are supported by backupOffline()
evalJS, evaluates JavaScript codes
createDomain, creates a domain
getDomain, gets a domain
dropDomain, drops a domain
isDomainExist, checks whether a domain exists or not
listDomain, enumerates and lists the domains

```

#### 2. New APIs in DBCollection Class:

```

alterCollection, alters the attributes of a collection

```

```
setMainKeys, sets the main keys of a collection. This API only works with save(), the main
keys set by it have no effects on other APIs
save, uses the default main key "-id" or other specified main keys to insert one or more
records
```

3. A Domain Class is added to support the domain-related operations

4. New APIs in SequoiadbDatasource Class:

```
SequoiadbDatasource, a new constructor supports multiple connecting addresses in order to
realize a better load balancing
getIdleConnNum, gets the number of currently idle connections
getUsedConnNum, gets the number of currently used connections
getNormalAddrNum, gets the number of currently normal connections
getAbnormalAddrNum, gets the number of currently abnormal connections
```

5. New APIs in SequoiadbOption class:

```
setRecaptureConnPeriod, sets the recapture connection period
getRecaptureConnPeriod, gets the recapture connection period
```

Version1.6

1. Add a class Node to replace the original class ReplicaNode. Class ReplicaNode and methods associated with it will be abandoned in the version 2.x.

For details, please see the relevant API.

## PHP Driver

This section introduce PHP driver.

[PHP Dirver](#)

[PHP Environment to Build](#)

[PHP Development Foundation](#)

[SQL to sequoiadb shell to PHP](#)

[PHP API](#)

PHP Dirver

### Outline

SequoiaDB PHP driver provides a PHP interface to database operations, and cluster operations. Database operations, including connectivity, users create a database to delete, add or delete data to the inverstigation, remove the index is created, the snapshot acquisition and replacement, as well as the creation of collections and collection space deletion operations. Cluster operations including operations management and data nodes partition groups, such as starting, stopping partition groups, start ,stop data nodes, obtain data from the master node, a collection of films.

There are two types of driving PHP instance. A database cooperation, another operation for the cluster.

- Examples of database operations

SequoiaDB stored in the database is divided into three levels:

- 1) Database
- 2) collection of space
- 3) collection

There fore, in the operation of the database, the available three classes to represent each connection, a collection of space, collection instance, another instance of a class represents a cursor:

SequoiaDB	Database class
SequoiaCS	collection space class
SequoiaCollection	collections
SequoiaCursor	cursor calss

PHP driver needs to use a different instance to operate. For example, the operation requires a cursor to read the data instances, and create a table space is required database instance.

- Examples of cluster operation

SequoiaDB cluster operations are divided into two levels: 1) partition group 2) data nodes



Note:

Partition group package of three types: Coordination partition group, cataloging partition group, the data partition group.

Partition group instance and instance data node instances can be expressed in two classes.

SequoiaGroup	partition group class	Partition group instance represents a separate partition group
SequoiaNode	Data node class	data node instance represents a single data node

And requires the use of cluster-related operations and data partition group node instance.

Examples SequoiaGroup for management partition group. Its operations include start, stop partition groups, access to state partition group node name information, the number of messages. Examples SequoiaNode for managing node. Its operations include start, stop specified node for the specified node instance, to obtain a master-slave node instance, access to data node address information.

### Error Messages

- A function call is successful return true (or integer 1), otherwise the return value is false (or integer 0).
- If the user needs to know the details of the error message, you can call `getError()` to get the error message, if there is no error, then output "No error message"

### PHP Environment to Build

#### Download the Driver Development Kit

Download the SequoiaDB Driver Development Kit for the corresponding system from <http://www.sequoiadb.com>. Extract the Driver Development Kit from `driver/lib/phpliblibsdbphp-x.x.x.so` file (x.x.x is the version number, please choose according to the PHP version, the first two digits need to exactly the same as the version, the third digit need to be less than the PHP version).

### Development Configuration

- Linux

Preparation: Install apache and PHP environment, PHP should be Version 5.3.3 or higher.

Configuration Steps:

1. Open `/etc/php5/apache2/php.ini` file.
2. Add the following line in the [PHP] configuration lines:

```
extension=<PATH>/liblibsdbphp-x.x.x.so
```

PATH is the path stores the `phpliblibsdbphp-x.x.x.so` file.

3. Save and close the file.

#### 4. Restart apache2 service

```
service apache2 restart (SUSE/Redhat) or service httpd restart (CentOS)
```

5. Write a PHP test script includes the following line, save it as test.php file, and save it under the Web service directory:

```
<?php phpinfo(); ?>
```

6. Open <http://localhost/test.php> using the explorer, check whether the page includes a SequoiaDB module.

- Windows

The Windows Driver Development Kit is not supplied currently.

### PHP Development Foundation

Here's how to use PHP database-driven interface written using SepuoiaDB program. For simplicity, this example is not entirely complete code, only play an exemplary role. Available to the installation directory /client/samples/php obtained under the corresponding complete code. [More View PHP API](#)

### Data Manipulation

- Connect to the database

```
//Create SequoiaDB object
$db = new Sequoiadb();
//Connect to database
$array = $db -> connect("localhost:11810");
//Test connection results returned by default php array type, the data is array(0)
{"errno"=>0}
//If errno is 0, then the connection is successful
if($array['errno'] !=0 )
{
    exit();
}
```

- Select the collection space

```
//Select the name of "foo" is a collection of space, if does not exist, it is automatically
created
//Return SequoiaCS object
$cs = $db -> selectCs("foo");
//Test results, if successful, returns an object, failed to return NULL
if( empty($cs) )
{
    exit();
}
```

- Choose a collection

```
//Select the name of "big" collection, and of not, is automatically created
//Return SequoiaCollection object
$c1 = $cs -> selectCollection("big");
//Test results, if successful, returns an object, failed to return NULL
if( empty($c1) )
{
    exit();
}
```

- Insert

```
//Insert json
$arr = $c1 -> insert(" {test:1} ");
//Test results
if($array['errno'] !=0 )
{
    exit();
}
```



```

}
//Insert array
$arr = $c1 -> insert(array("test">2));
//Test results
if($arr['errno'] !=0 )
{
    exit();
}

```

- Inquiry

```

//Set all the records in the query
//Return SequoiaCursor object
$cursor = $c1 -> find();
//Loop through all records
while($record = $cursor -> getNext())
{
    var_dump($record);
}

```

- Update

```

//Modify the collection of more than a record, the value of the field test was revised to 0
$arr = $c1 -> update(" {$set: {test:0}} ");
//Test results
if($arr['errno'] !=0 )
{
    exit();
}

```

- Delete

```

//Delete all records collection
$arr = $c1 -> remove();
//Test results
if($arr['errno'] !=0 )
{
    exit();
}

```

## Cluster operation

- Select Group

```

//Select the name of "group" of the group, if you do not exist, it is automatically created
//Return SequoiaShard object
$group = $db -> selectGroup("group");
//Test results, if successful, returns an object, failed to return NULL
if( empty($group) )
{
    exit();
}

```

- Boot partition group

```

//boot partition group for the first time will automatically activate
//Return operation information
$arr = $group -> start() ;
//Check the results
If ( $arr['errno'] != 0 )
{
    Exit() ;
}

```

- Select the node

```

//Get the name, of "node" node
//Return SequoiaNode object
$node = $group -> getNode( 'node' ) ;
//Check whether the object is empty
If ( empty( $node ) )

```

```
{
    Exit () ;
}
```

### SQL to sequoiadb shell to PHP

The query in Sequoiadb is in the format of json(bson). The following table shows examples comparing sql statements, sequoiadb shell statements and sequoiadb PHP driver application statements.

SQL	sequoiadb shell	php Driver
insert into students(a,b) values(1,-1)	db.foo.bar.insert({a:1,b:-1})	\$bar->insert( "{a:1,b:-1}" )
select a,b from students	db.foo.bar.find(null,{a:"",b:""})	\$bar->find(NULL,'{a:"",b:""}')
select * from students	db.foo.bar.find()	\$bar->find()
select * from students where age=20	db.foo.bar.find({age:20})	\$bar->find("{age:20}")
select * from students where age=20 order by name	db.foo.bar.find({age:20}).sort({name:1})	\$bar->find("{age:20}",NULL,"{name:1}")
select * from students where age>20 and age<30	db.foo.bar.find({age:{>:20,<:30}})	\$bar->find("{age:{>:20,<:30}}")
create index testIndex on students(name)	db.foo.bar.createIndex("testIndex", {name:1},false)	\$bar->createIndex("{name:1}", "testIndex",false)
select * from students limit 20 skip 10	db.foo.bar.find().limit(20).skip(10)	\$bar->find(NULL,NULL,NULL,NULL,10,20)
select count(*) from students where age>20	db.foo.bar.find({age:{>:20}}).count()	\$bar->count("{age:{>:20}}")
update students set a=a+2 where b=-1	db.foo.bar.update({\$set:{a:2}},{b:-1})	\$bar->update("{\$set:{a:2}}","{b:-1}")
delete from students where a=1	db.foo.bar.remove({a:1})	\$bar->remove("{a:1}")

### PHP API

This part is related to the PHP API document.

### PHP API

## C# Driver

this section introduce C# driver

### C# Driver

### C# Environment to Build

### C# Development Foundation

### SQL to sequoiadb shell to C#

### C# API

### C# Driver

### Outline

SequoiaDB C# driver provides an interface database operations, and cluster operations. Database operations, including connectivity, users create a database to delete, add or delete data to the investigation, remove the index is created, the snapshot acquisition and replacement, as well as the creation of collections and collection space deletion operations. Cluster operations including operations management and data nodes partition groups, such as starting, stopping partition groups, start, stop data nodes, obtain data from the master node, a collection off films.

### c# class instance

C# class instance two drives. A database operation, another operation for the cluster.

- Examples of database operations

SequoiaDB stored in the database is divided into three levels:

- 1) database
- 2) collection of space

### 3) collection

Therefore, database operations, respectively, connected by three class represents a collection space, a collection of instances of a class represents another cursor.

SequoiaDB	Database instance	Represents a single database connection
CollectionSpace	Examples of the collection space	Represents a separate set of space
DBCollection	Collection instance	A separate set of representatives
DBCursor	Cursor instance	Represents a query result sets generated

c# driver needs to use a different instance to operate. For example, the operation requires a cursor to read the data instances, and create a table space is required database instance.

- Wxamples of cluster operation

SequoiaDB database cluseer operations are divided into two levels:

- 1) partition group
- 2) Data Node



Note: partition group contains three types: Coordination partition group, cataloging partition group, the data partition group.

Partition group instance and instance data node instances can be expressed in two classes.

Shard	Partition group class	Partition group instance represents a separate partition group
Node	Data node class	Data node instance represents a single data node

Undoubtedly related to the cluster operation requires the use of partition groups and data node instance.

Examples Shard for management partition group. Its operations include start, stop partition groups, access to state partition group node name information, the number of messages.

Examples Node for managing node. Its operations include start, stop specified node for the specified node instance, to obtain a master-slave node instance, access to data node address information.

### Thread safety

For each connection, which produces a set of space, a collection of utilities a socket. Therefore, in a multi-threaded system, you must ensure that each thread is not the same tim for the same socket, send or receive data at the same time. Generally it is not recommended to use multiple threads to operate together with its other instance of a connection instance generated. If each thread uses its own connection as well as other examples of instances generated, you can ensure thread safety.

### Error Messages

Each interface will throw SequoiaDB.BaseException and System.Exception exception, corresponding to the exception information and client-side local database engine to return exception information, which can be obtained by exception information BaseException the exception class ErrorCode, ErrorType and Message properties get.

### C# Environment to Build

#### Get Driver Development Kit

Form <http://www.sequoiadb.com> download the corresponding operatin system version SequoiaDB Driver Development Kit, extract the driver development kit, get sequoiadb.dll link library form the driver/ CSharp/ directory, then refer to the link library in Visual Studio or specify a reference on the command line to compile the libraries, such as "csc /target:exe /reference:sequoiadb.dll Find.cs Common.cs", you

can use the relevant API. Samples in the installation directory ~~W~~C# directory you can find a complete sample C# driven.

### BSON library API

SequoiaDB using C# driver for MongoDB BSON library, detailing MongoDB can refer to the official documentation: <http://docs.mongodb.org/ecosystem/tutorial/use-csharp-driver/#the-bson-library>

### Visual Studio versions supported

The current version of C# driver can be used in the following versions of Visual Studio

Visual Studio 2008

Visual Studio 2010

### .NET Framework version support

The current version of the driver in C# .NET Framework3.5 generated, can be used in the following versions .NET Framework in

.NET Framework 3.5

.NET Framework 4.0

### C# Development Foundation

Here's how to use the driver interface written in C# using SequoiaDB database program. This document describes a simple C# sample SequoiaDB database-driver, detailed specification can refer to the official use of [C# API documentation](#).

### Namespace

Before using C# driver related API, you must add the following using declaration in source code:

```
using SequoiaDB;
```

```
using SequoiaDB.Bson;
```

### Data Manipulation

- Connect to the database and authentication

If the database does not create a user, you can anonymously connect to the database:

```
string addr = "127.0.0.1:11810";
Sequoiadb sdb = new Sequoiadb(addr);
try
{
    sdb.Connect();
}
catch (BaseException e)
{
    Console.WriteLine("ErrorCode: {0}, ErrorType: {1} ", e.ErrorCode, e.ErrorType);
    Console.WriteLine(e.Message);
}
catch (System.Exception e)
{
    Console.WriteLine(e.StackTrace);
}
```

Otherwise, when the connection must specify a user name and password:

```
string addr = "127.0.0.1:11810";
Sequoiadb sdb = new Sequoiadb(addr);
```

```

try
{
    sdb.Connect("testusr", "testpwd");
}
catch (BaseException e)
{
    Console.WriteLine("ErrorCode: {0}, ErrorType: {1} ", e.ErrorCode, e.ErrorType);
    Console.WriteLine(e.Message);
}
catch (System.Exception e)
{
    Console.WriteLine(e.StackTrace);
}

```

Here are the exception information try and catch blocks, all operations following will throw the same exception information, it is no longer relevant given the try and catch blocks.

- Disconnect from the database

```

// do not forget to disconnect from sdb
sdb.Disconnect();

```

- Get or create a collection space and collections

By name, to give the corresponding CollectionSpace, if not, create:

```

// create collectionspace, if collectionspace exists get it
string csName = "TestCS";
CollectionSpace cs = sdb.GetCollectionSpace(csName);
if (cs == null)
cs = sdb.CreateCollectionSpace(csName);
// or sdb.CreateCollectionSpace(csName, pageSize), need to specify the pageSize

```

By name, to give the corresponding Collection, and if not, create:

```

// create collection, if collection exists get it
string clName = "TestCL";
DBCollection dbc = cs.GetCollection(clName);
if (dbc == null)
dbc = cs.CreateCollection(clName);
//or cs.CreateCollection(collectionName, options), create collection with some options

```

- Insert operation for Collection

Need to insert data BsonDocument create and insert:

```

BsonDocument insertor = new BsonDocument();
string date = DateTime.Now.ToString();
insertor.Add("operation", "Insert");
insertor.Add("date", date);
ObjectId id = dbc.Insert(insertor);

```

Of course, BsonDocument BsonDocument can also be nested object, And you can also directly a complete new BsonDocument, without going through the Add method:

```

BsonDocument insertor = new BsonDocument
{
    {"FirstName", "John"},
    {"LastName", "Smith"},
    {"Age", 50},
    {"id", i},
    {"Address",
        new BsonDocument
        {
            {"StreetAddress", "212ndStreet"},
            {"City", "new York"},
            {"State", "NY"},
            {"PostalCode", "10021"}
        }
    }
}

```

```

    },
    {"PhoneNumber",
     new BsonDocument
     {
         {"Type", "Home"},
         {"Number", "212555-1234"}
     }
    }
};
Insert the number of data:
//bulkinsert
List<BsonDocument> insertor=new List <BsonDocument> ();
for (inti=0; i<10; i++)
{
    BsonDocumentobj=new BsonDocument ();
    obj.Add ("operation", "BulkInsert");
    obj.Add ("date", DateTime.Now.ToString());
    insertor.Add (obj);
}
dbc.BulkInsert (insertor, 0);

```

- **Index-related operations**

Create an index:

```

//createindexkey, indexonattribute' Id' byASC (1) /DESC (-1)
BsonDocument key = new BsonDocument ();
key.Add ("id", 1);
string name = "index name";
bool isUnique = true;
bool isEnforced = true;
dbc.CreateIndex (name, key, isUnique, isEnforced);

```

Delete Index:

```

string name = "index name";
dbc.DropIndex (name);

```

- **Query operation**

Query, you need to use a cursor to traverse the query results, but can first get the index of the current Collection, if not empty, as the development of an access plan (hint) for inquiries:

```

DBCursor icursor = dbc.GetIndex (name);
BsonDocument index = icursor.Current ();

```

BsonDocument objects are used to construct the corresponding queries, including: Query matching rules (matcher, include the appropriate query), domain selection (selector), sorting rules (orderBy, in increasing order or descending), to develop access plans (hint), skip record number (0), the number of records returned (-1: return all data). After the inquiry, to obtain the corresponding Cursor, for the results obtained by traversing the query:

```

BsonDocument matcher = new BsonDocument ();
BsonDocument conditon = new BsonDocument ();
conditon.Add ("$gte", 0);
conditon.Add ("$lte", 9);
matcher.Add ("id", conditon);
BsonDocument selector = new BsonDocument ();
selector.Add ("id", null);
selector.Add ("Age", null);
BsonDocument orderBy = new BsonDocument ();
orderBy.Add ("id", -1);
BsonDocument hint = null;
if (index != null)
    hint = index;
else
    hint = new BsonDocument ();
DBCursor cursor = dbc.Query (matcher, selector, orderBy, hint, 0, -1);

```

Use DBCursor cursor to traverse:

```
while (cursor.Next() != null)
    Console.WriteLine(cursor.Current());
```

- Delete operation

Construct the corresponding BsonDocument object, set the conditions for deletion:

```
//createthedeletecondition
BsonDocument drop = new BsonDocument();
drop.Add("Last Name", "Smith");
coll.Delete(drop);
```

- Update operation

Construct the corresponding BsonDocument object used to set the update condition, you can also create DBQuery object encapsulates all of the query or update rules:

```
DBQuery query = new DBQuery();
BsonDocument updater = new BsonDocument();
BsonDocument matcher = new BsonDocument();
BsonDocument modifier = new BsonDocument();
updater.Add("Age", 25);
modifier.Add("$set", updater);
matcher.Add("First Name", "John");
query.Matcher = matcher;
query.Modifier = modifier;
dbc.Update(query);
```

Update operation, if the conditions are not met matcher, then insert this record:

```
dbc.Upsert(query);
```

## SQL to sequoiadb shell to C#

The query in Sequoiadb is in the format of json(bson). The following table shows examples comparing sql statements, sequoiadb shell statements and sequoiadb C# driver application statements.

SQL	sequoiadb shell	C# Driver
insert into students(a,b) values(1,-1)	db.foo.bar.insert({a:1,b:-1})	bar.insert("{\"a\":1,\"b\":-1}")
select a,b from students	db.foo.bar.find(null,{a:"",b:""})	bar.query("", "{a:'',b:''}", "", "")
select * from students	db.foo.bar.find()	bar.query()
select * from students where age=20	db.foo.bar.find({age:20})	bar.query("{\"age\":20}", "", "", "")
select * from students where age=20 order by name	db.foo.bar.find({age:20}).sort({name:1})	bar.query("{\"age\":20}", "", "{\"name\":1}", "")
select * from students where age>20 and age<30	db.foo.bar.find({age:{>:20,<:30}})	bar.query("{\"age\":{\">\":20,\"<\":30}}", "", "", "")
create index testIndex on students(name)	db.foo.bar.createIndex("testIndex", {name:1},false)	bar.createIndex("testIndex", "{name:1}", false, false)
select * from students limit 20 skip 10	db.foo.bar.find().limit(20).skip(10)	bar.query("", "", "", "", 10, 20)
select count(*) from students where age>20	db.foo.bar.find({age:{>:20}}).count()	bar.getCount("{\"age\":{\">\":20}}")
update students set a=a+2 where b=-1	db.foo.bar.update({\$set:{a:2}},{b:-1})	bar.update("{\"b\":-1}", "{\$inc\":{\"a\":2}}", "")
delete from students where a=1	db.foo.bar.remove({a:1})	bar.delete("{\"a\":1}")

## C# API

This part is related to C# API document.

## C# API

## BSON Interface to C

BSON the binary format of JSON. Through recording every object or element, nested element and the type and length of array, elements can be rapidly and efficiently found. So in C applications, we use BSON interface provided by authority to store data. Different from common JSON, BSON supports more data types in order to meet the all kinds of requirements of C. SequoiaDB support data types including 8-byte double float, string, nested object, nested array, object ID (Each record in database has a unique ID),

bool, data, NULL, regex, 4-byte integer (INT), timeline, 8-byte integer, etc. The definition of these types can be found in `bson_type` in a file called `bson.h`.

When users use `bson` object, they mainly create and read object .

### Create Object

In general, the process of generation of a BSON object contains 3 steps:

- 1)Create object (`bson_create ; bson_init`)
- 2)Use object
- 3)Clear object (`bson_dispose ; bson_destory`)

- Create a simple BSON object {age:20}.

```
INT32 rc = SDB_OK;
bson obj;
bson_init(&obj);
bson_appent_int(obj, "age", 20);
if ( bson_finish(obj) != SDB_OK )
printf("Error. ") ;
bson_destory(obj);
```

- Create a complex BSON object

```
/* Create a object contianing {name:"tom",colors:["red","blue","green"], address:
{city:"Toronto, province: "Ontario"}} */
bson_iterator bi ;
bson *newobj = bson_create () ;
bson_append_string ( newobj, "name", "tom" ) ;
bson_append_start_object ( newobj, "address" ) ;
bson_append_string ( newobj, "city", "Toronto" ) ;
bson_append_string ( newobj, "provice", "Ontario" ) ;
bson_append_start_array(newobj, "colors");
bson_appent_string(newobj, "0", "red");
bson_appent_string(newobj, "1", "blue");
bson_appent_string(newobj, "2", "green");
bson_append_finish_object ( newobj ) ;
if( bson_finish ( newobj ) != BSON_OK )
printf("Error. ") ;
```

### Read Object

We use `bson_iterator` to read BSON object. In a complete example, the function `bson_print_raw` can be used to rad data. We initiate the BSON object and go through each element with the function called `bson_iterator_next`.

```
bson_iterator i[1] ;
bson_type type ;
const char * key;

bson_iterator_init(i, newobj) ;

type = bson_iterator_next (i);
key = bson_iterator_key (i);

printf( "Type: %d, Key: %s\n", type, key) ;
```

For every `bson_iterator`, the function `bson_iterator_type` can return the type of it. Functions like `bson_iterator_string` return values in corresponding type.

```
printf( "Value: %s, bson_iterator_string(i)) ;
```



- When we go through consecutive BSON object elements, we can directly get the name of a element with the function `bson_find` . For example, we can ge the "name" value of a element in this sample.

```
bson_iterator i[1] , sub[i] ;
bson_type type ;

bson_find ( i, newobj, "name" )
```

- Read array element or nested objects,as "address" is a nested object,it need to special iterate.Getting value of "address" at first,then init a new BSON iterator:

```
type = bson_find(i, newobj, "address");
bson_iterator_subiterator(i, sub);
```

The method `bson_iterator_subiterator` inits iterator sub,and points to the beginning of sub-objects,then it can iterate all the elements in the sub until the end of sub-objects.

## BSON Interface to C++

C++ BSON main categories

C++ BSON uses four categories:

`bson::BSONObj`: Creating `BSONObj` object.

`bson::BSONElement`: `BSONObj` object consists `BSONElement` object, namely `BSONElement` object field or element `BSONObj` object, which is the key right.

`bson::BSONObjBuilder` : `BSONObjBuilder` `BSONObj` used to instantiate the object.

`bson::BSONObjIterator` : `BSONObjIterator` used to rtaverse `BSONObj` object elements.

Bson namespace type of these classes are defined as follows:

```
typedef bson::BSONElement be;
```

```
typedef bson::BSONObj bo;
```

```
typedef bson::BSONObjBuilder bob;
```

In addition, you can use the `bo::iterator` instead of PHP development environment to build `BSONObjIterator`.

Create an objcet

The following describes how to create a simple to use CPP BSON instance. For details ,see [C++ BSON API](#)

- Use `BSONObject`, `BSONObjBuilder` builder create an object.

```
#include "client.hpp"
...
using namespace bson ;
BSONObj obj ;
BSONObjBuilder b ;

b.append("name", "sam") ;
b.append("age", "24") ;
obj = b.obj() ;
or
obj = BSONObjBuilder().genOID().append("name", "sam").append("age", 24).obj() ;
```

Further, the method may be used to establish the data flow `BSONObj` lbject.

```
BSONObj obj ;
BSONObjBuilder b ;
b<<"name"<<"sam"<<"age"<<"24" ;
obj = b.obj() ;
```

- Create an object using a macro `BSON`

C++ BSON defined also defines a BSON macro, you can use it to build BSONObj objects quickly.

```
BSONObj obj ;
// int
obj = BSON( "a"<<1 ) ;
// float
obj = BSON( "b"<<3.14159265359 ) ;
// string
obj = BSON( "foo"<<"bar" ) ;
// OID
obj = BSON( GENOID ) ;
// bool
obj = BSON( "flag"<<true"ret"<<false ) ;
// object
obj = BSON( "d"<<BSON("e"<<"hi!") ) ;
// array
obj = BSON( "phone" << BSON_ARRAY( "13800138123" << "13800138124" ) ) ;
// others, less then, greater then, etc
obj = BSON( "g"<<LT<<99 ) ;
```

- Interface to create an object using fromjson.

In addition, you can use fromjson.hpp the fromjson() to convert a string into BSONObj json object.

```
string s( " {name: \"sam\"} " ) ;
fromjson ( s, obj ) ;
or
const char *r = " {
    firstName: \"Sam\", \
    lastName: \"Smith\", age: 25, id: \"count\", \
    address: {streetAddress: \"25 3rdStreet\", \
    city: \"NewYork\", state: \"NY\", postalCode: \"10021\"}, \
    phoneNumber: [ {type: \"home\", number: \"212555-1234\"} ] } " ;
fromjson ( r, obj ) ;
```

## C BSON API

This part is related to C BSON API documentation.

[C BSON API](#)

## C++ BSON API

This part is related to C++ BSON API documentation.

[C++ BSON API](#)

## SequoiaDB Cluster Manager

---

### sdbcm overview

SequoiaDB Cluster Manager is a daemon process , in Windows, it disposes the system background in the way of service. All of the Cluster Manager operations must have the participation of sdbcm in Sequoiadb, at present, each physical machine can only start a sdbcm process, responsible for executing the remote commands and Monitoring the local Sequoiadb 。 sdbcm mainly has two big functions :

- Remote starting、closing、creating and modifying nodes: Through Sequoiadb client or driving to connect to the database, then we can execute the operations of starting、closing、creating and modifying nodes, those operations send remote commands to sdbcm that the physical machine specified, and obtaining performance results.
- local monitoring : for the starting nodes by sdbcm,will maintain a table about nodes, it save all the service name of the local nodes and starting information, such as starting time、running status

etc. If a node terminate in normal case, such as the process is forced to terminate、 engine quit unexpectedly etc., then sdbcm will try restart this node.

#### sdbcm operator

- conf files

in the database installtion directory, hava a conf file named sdbcm.conf, this file gives the conf information of starting sdbcm, as follows:

param	description	example
defaultPort	the default monitoring port of sdbcm	defaultPort=11790
<hostname>_Port	monitoring port of sdbcm in physical host.if it can't find the corresponding param of host in the conf file,sdbcm will start with defaultPort; if defaultPort is also not exist ,then sdbcm will start the default port 11790	<hostname>_Port=11790
RestartCount	number of restarts. That is define the max number of restarting nodes for sdbcm. If the param doesn't exist,then default set to -1,that is constantly restart.	RestartCount=5
RestartInterval	restart interval, that is define the max restart interval of sdbcm,the unit is minutes. This param combine with RestartCount define the max number of restarting nodes for sdbcm in the restart interval, when beyond,no longer restart. If the param doesn't exist,then default set to 0, that is doesn'tconsider the restart interval .	RestartInterval=0

- start sdbcm

running sdbcmart commands can start sdbcm.

- close sdbcm

running sdbcmstop command can close sdbcm.

## Reference

It introduces javascript methods and operator, vocabulary, exception information and name limits in SequoiaDB database.

### SequoiaDB JavaScript Method List

Sequoiadb is a kind of file-based non-relation database. It usesJSON data module.

- If users want to match elements nested in an object. They can use "." ("0x2E" in ASCII) to connect parent element and child element. For example, in this object:

```
{ name: "tom", address: { street: { street1: "1024 Wall Street", street2: "University Drive"}, zipcode: 100000 } }
```

If users want to match the element "street" in the object "street" in the object "address", they can get it in this way:

```
{ "address.street.street1": { $exists : 1 } }
```

- If there is an array in an object, users can get the 1st, 2nd, and 3rd element by "0", "1" and "2" respectively. For example, this object contains an array:

```
{ name: "tom", phone: [ "13175824", "1528345" ] }
```

Users can get the 1st element in the array in this way:

```
{ "phone.0 ": "13175824" }
```

#### Database Operation Methods

Method Name	Description	Sample
<a href="#">db.createCS()</a>	Create a collection space	db.createCS("foo")
<a href="#">db.dropCS()</a>	Delete a collection space	db.dropCS("foo")
<a href="#">db.getCS()</a>	Get the quotation of a collection space	db.getCS("foo")
<a href="#">db.createRG()</a>	Create a replset	db.createRG("rg")
<a href="#">db.removeRG()</a>	Remove a replset	db.removeRG("rg")
<a href="#">db.getRG()</a>	Get the quotation of a replset	db.getRG("rg")
<a href="#">db.list()</a>	Enumerate lists	db.list()
<a href="#">db.listCollectionSpaces()</a>	Enumerate collection spaces	db.listCollectionSpaces()
<a href="#">db.listCollections()</a>	Enumerate collections	db.listCollections()
<a href="#">db.listReplicaGroups()</a>	Enumerate replsets	db.listReplicaGroups()
<a href="#">db.snapshot()</a>	Enumerate snapshots	db.snapshot(0)
<a href="#">db.resetSnapshot()</a>	Reset a snapshot	db.resetSnapshot(0)
<a href="#">db.startRG()</a>	Start a replset	db.startRG("rgName")
<a href="#">db.createUsr()</a>	Create a database user	db.createUsr("root","admin")
<a href="#">db.dropUsr()</a>	Delete a database user	db.dropUsr("root","admin")
<a href="#">db.transBegin()</a>	Open the transaction	db.transBegin()
<a href="#">db.transCommit()</a>	Commit the transaction	db.transCommit()
<a href="#">db.transRollback()</a>	Transaction rollback	db.transRollback()
<a href="#">db.createProcedure()</a>	Create a stored procedure functions	db.createProcedure(function sum(i,j){return i+j;})

Method Name	Description	Sample
<a href="#">db.removeProcedure()</a>	Delete stored procedure function that already exists	<code>db.removeProcedure("sum")</code>
<a href="#">db.listProcedures()</a>	Enumeration stored procedure functions	<code>db.listProcedures()</code>
<a href="#">db.eval()</a>	Call a stored procedure or function directly using javaScript statement	<code>db.eval('db.foo.bar')</code>

### Collection Space Method

Method Name	Description	Sample
<a href="#">db.collectionspace.createCL()</a>	Create a collection	<code>db.foo.createCL("bar")</code>
<a href="#">db.collectionspace.dropCL()</a>	Delete a collection	<code>db.foo.dropCL("bar")</code>
<a href="#">db.collectionspace.getCL()</a>	Get the quotation of a collection	<code>db.foo.getCL("bar")</code>

### Collection Method

Method Name	Description	Sample
<a href="#">db.collectionspace.collection.insert()</a>	Insert a record into a collection	<code>db.foo.bar.insert({name:"Tom",age:20,phone:[123,345]})</code>
<a href="#">db.collectionspace.collection.count()</a>	Count the records in a collection	<code>db.foo.bar.count({age:{&gt;20}})</code>
<a href="#">db.collectionspace.collection.find()</a>	Find records	<code>db.foo.bar.find()</code>
<a href="#">db.collectionspace.collection.aggregate()</a>	Gather records	<code>db.foo.bar.aggregate({\$project:{name:1,age:1}},{\$group:{_id:"\$sex"}})</code>
<a href="#">db.collectionspace.collection.remove()</a>	Delete records	<code>db.foo.bar.remove()</code>
<a href="#">db.collectionspace.collection.createIndex()</a>	Create an index	<code>db.foo.bar.createIndex("myIndex",{age:-1},false)</code>
<a href="#">db.collectionspace.collection.dropIndex()</a>	Delete an index	<code>db.foo.bar.dropIndex("myIndex")</code>
<a href="#">db.collectionspace.collection.getIndex()</a>	Get the quotation of an index	<code>db.foo.bar.getIndex("myIndex")</code>
<a href="#">db.collectionspace.collection.listIndexes()</a>	Enumerate indexes	<code>db.foo.bar.listIndexes()</code>
<a href="#">db.collectionspace.collection.update()</a>	Update records	<code>db.foo.bar.update({\$set:{age:25}},{age:{&lt;=20}},{"":"myIndex"})</code>
<a href="#">db.collectionspace.collection.upset()</a>	Update records	<code>db.foo.bar.update({\$set:{age:25}},{a:1},{"":"myIndex"})</code>
<a href="#">db.collectionspace.collection.split()</a>	Split a collection	<code>db.foo.bar.split("sourceRG","targetRG",{age:20})</code>

### Cluster Method

Method Name	Description	Sample
<a href="#">rg.start()</a>	Start replset	<code>rg.start()</code>
<a href="#">rg.getDetail()</a>	Get information of replset	<code>rg.getDetail()</code>
<a href="#">rg.createNode()</a>	Create node	<code>rg.createNode(&lt;hostname&gt;,&lt;port&gt;,&lt;path&gt;)</code>
<a href="#">rg.removeNode()</a>	Delete node	<code>rg.removeNode(&lt;host&gt;,&lt;service&gt;[,&lt;config&gt;])</code>
<a href="#">rg.getMaster()</a>	Get information of master node	<code>rg.getMaster()</code>
<a href="#">rg.getSlave()</a>	Get information of slave node	<code>rg.getSlave()</code>
<a href="#">rg.getNode()</a>	Get information of all nodes	<code>rg.getNode()</code>
<a href="#">rg.stop()</a>	Stop a replset	<code>rg.stop()</code>

### Node Method

Method Name	Description	Sample
<a href="#">node.start()</a>	Start a node	<code>node.start()</code>

Method Name	Description	Sample
<a href="#">node.connect()</a>	Get the hostname and port of a connected node	<code>node.connect()</code>
<a href="#">node.getHostName()</a>	Get the host name of a node	<code>node.getHostName()</code>
<a href="#">node.getServiceName()</a>	Get the server name of a node	<code>node.getServiceName()</code>
<a href="#">node.getNodeDetail()</a>	Get information of a node	<code>node.getNodeDetail()</code>
<a href="#">node.stop()</a>	Stop a node	<code>node.stop()</code>

### Cursor Method

Method Name	Description	Sample
<a href="#">cursor.sort()</a>	Sort result set according to the specified field	<code>db.foo.bar.find().sort({age:1})</code>
<a href="#">cursor.hint()</a>	Go through result set according to the specified index	<code>db.foo.bar.find().hint({"":"Index"})</code>
<a href="#">cursor.limit()</a>	Specify the amount of records to be returned in the result set	<code>db.foo.bar.find().limit(10)</code>
<a href="#">cursor.skip()</a>	Specify the first record to be returned in the result set	<code>db.foo.bar.find().skip(10)</code>
<a href="#">cursor.current()</a>	Return the record that the current cursor currently points to	<code>db.foo.bar.find().current()</code>
<a href="#">cursor.next()</a>	Return the next record in the cursor	<code>db.foo.bar.find().next()</code>
<a href="#">cursor.size()</a>	Return the distance from the record that the current cursor points to to the final record	<code>db.foo.bar.find().size()</code>
<a href="#">cursor.toArray()</a>	Return result set in the format of array	<code>db.foo.bar.find().toArray()</code>

## db.backupOffline()

NAME

backupOffline - Backup Database.

SYNOPSIS

```
db.backupOffline([options])
```

CATEGORY

Sequoiadb

DESCRIPTION

Backup Database.

options (json object)

Specify the name of the backup, the replica group, and number of copies, etc.

GroupID

Specify the ID of the replica group need to be backup. Backup all replica groups in default.

GroupID:1000 or GroupID:[1000, 1001]

GroupName

Specify the name of the replica group need to backup. Backup all replica groups in default.

GroupName:"data1" or GroupName:["data1", "data2"]

**Name**

Backup name, the default name is in the "YYYY-MM-DD-HH:mm:ss" time format.

Name:"backup-2014-1-1"

**Path**

Backup path, the default path is the one specified in the configuration. The path supports wildcards(%g%G: group name, %h%H:host name, %s%S:service name).

Path:"/opt/sequoiadb/backup/%g"

**IsSubDir**

Whether the path configured in the above Path parameter is a subdirectory of the backup path in the configuration. False in default.

IsSubDir:false

**Prefix**

Backup prefix, support wildcards (%g,%G,%h,%H,%s,%S), The default is empty.

Prefix:"%g\_bk\_"

**EnableDataDir**

Whether to enable the date-subdirectory function or not. It will automatically create subdirectories with names in "YYYY-MM-DD" format according to the current date if the function is enabled. False in default.

EnableDataDir:false

**Description**

Backup description

Description:"First backup"

**EnsureInc**

Whether to enable incremental backup or not. False in default.

EnsureInc:false

**OverWrite**

Whether to overwrite the backups with the same name. False in default.

OverWrite:false

**Examples**

Back up the entire database.

```
db.backupOffline({Name: "FullBackup1"})
```

**db.cancelTask()****NAME**

cancelTask - Cancel the task.

**SYNOPSIS**

```
db.cancelTask(<id>,[isAsync])
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

Cancel the task.

id (Integer)

Task ID

isAsync (Boolean)

Whether to cancel the task asynchronously or not.

**Examples**

Stop a split task asynchronously.

```
var taskId1 = db.test.test.splitAsync("db1", "db2", 50);
db.cancelTask( taskId1, true )
```

**db.createCataRG()****NAME**

createCataRG - Create a new catalog replica group.

**SYNOPSIS**

```
db.createCataRG(<host>,<service>,<dbpath>,[config])
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

Create a new catalog replica group, create and start a catalog node at the same time.

host (string)

Specify the host name of the catalog node.

service (int)

Specify the service port of the catalog node, please make sure this port and its three succeeding ones are unoccupied. For example, if 11800 is specified, please make sure all of 11800/11801/11802/11803 are unoccupied.

dbpath (string)

The path for the data files which is used to store the catalog data files. Please make sure that the data administrator users(created during installation, 'sdbadmin' in default) have the write permissions.

config (json)

Optional. Used to set up more detailed configurations. It must be in Json format. For more details please check the Database Configuration section, for example the parameter for configuring of log file size: {logfilesz:64}.

Format

createCataRG() method has four parameters: host, service, dbpath, and config. The value of 'host' and 'dbpath' should be a string, 'service' should be an int, and a Json Object for 'config', the format is:

```
{"<host name>",<port number>",<data file path>",[database configuration parameter object]}
```



**Note:**

- Please make sure to give proper permissions to the data-file storage path, if default installation was used, then give the 'sdbadmin' permission to the path.

**Examples**

Named: Create a catalog node group on sdbserver1 host, service port: 11800, data files path: /opt/sequoiadb/database/cata/11800

```
db.createCataRG("sdbserver1", 11800, "/opt/sequoiadb/database/cata/11800")
```

**db.createCS()****NAME**

createCS - Create a collection space in a database instance.

**SYNOPSIS**

```
db.createCS(<name>,[options])
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

Create a collection space in a database instance.

name (string)

Collection space name. Collection space name should be unique to each other in a database instance.

options (Json object)

Optional in a collection space.

PageSize

Size of a data page. The default value is 65536B.

PageSize:<int32>

Domain

Domain that collection space belongs to.

Domain:<string>

**Note:**

- The parameter "name" should not be a null string. It should not contain "." or "\$". The length of it should not be greater than 127B.
- Collection space names should be unique to each other in a database instance.
- When creating a collection space, users can specify the size of data page. It is unchangeable afterward. The default value of it is 65536B.
- PageSize can only be one of the following six: 0, 4096, 8192, 16384, 32768, and 65536. The default value is 0.
- Domain must already exist, and cannot be SYSDOMAIN.
- It is compatible with earlier version interface, and function db.createCS(<name>, [PageSize]) can also be used.

### Examples

Create a collection space named "foo" without specifying the size of data page, so its default page size will be 65536B.

```
db.createCS("foo")
```

Create a collection space named "foo", specify the size of the data page as 4096 and its domain as "mydomain".

```
db.createCS("foo", {PageSize: 4096, Domain: "mydomain"})
```

## db.createDomain()

### NAME

createDomain - Create a domain.

### SYNOPSIS

```
db.createDomain(<name>,<groups>,[options])
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Create a domain. A domain can contain multiple replica groups.

name (string)

Unique domain name in the database.

groups (Json Array)

Replica groups belong to the domain.

options (Json Object)

Configuration options for creating a domain with different given fields.

AutoSplit

When this field is set to be True, all the hash-partitioning collection spaces created in this domain will be automatically splitted into the replica groups it belong to.

AutoSplit:true|false

Note:



- AutoSplit can only apply to hash-partitioning collections.
- Cannot create collection spaces in an empty domain(domain with no replica groups).

### Examples

Create a domain which contains two replica groups.

```
db.createDomain('mydomain', ['datagroup1','datagroup2'])
```

Create a domain which contains two replica groups, and enables its auto-split function.

```
db.createDomain('mydomain', ['datagroup1','datagroup2'], {AutoSplit: true})
```

## db.createProcedure()

### NAME

createProcedure - Create a stored procedure in the database instance.

### SYNOPSIS

```
db.createProcedure(<code>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Create a stored procedure in the database instance.

code (Custom Functions)

Standard function definition, not a string type, quotation marks cannot be used when inputting parameters.

Format

createProcedure() has a parameter "code" which is a standard function definition.

```
createProcedure(<code>)
```

- Recommend to directly use the database instance initialized during the stored procedure, for example: `db.createProcedure( function getAll(){return db.foo.bar.find();} )` . But if authentication is required, the global database will fail. A new database instance need to be initialized.
- Database initialization is in the form of: `var db=new Sdb()`. If user name and password are needed, it will be: `var db=new Sdb('username','passwd')`. Note that the stored procedure can only run on its own connected database, it does not support remote connection methods with other databases. In the case that authentication is not required, `var db = new Sdb('hostname', 'servicename')` can still execute successfully, the returned database is still the local database.
- Database's role must be a coord node. Standalone mode cannot support procedure function.

### Function definition

- Function definition

1. Function must contain the function name. It cannot be used in the following form `function(x,y){return x+y;}`

2. Other functions or even inexistent functions can be called in the function definition. But it should be ensured that all the functions are existent during runtime.

3. The function name should be globally unique and overload is not supported.

4. Each function is available in the whole system. Delete a procedure without proper check may lead to run time exceptions of other people.

- Function's parameters

native type of JS

- Function's outputs

Functions for all standard output, standard error will be masked. And the uses of output commands are not recommended in the function definitions. Too much output may lead to a failure during the running of stored procedure.

- Function's returned values

Function's returned values can be any data type except for a database. For example: function getCL()  
{return db.foo.bar;}

### Examples

Create a sum function:

```
db.createProcedure(function sum(x,y) {return x+y;})
```

Once the function is created, db.listProcedures() can be used to check the information of the functions.

## db.createRG()

### NAME

createRG - Create a replica group.

### SYNOPSIS

```
db.createRG(<name>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Create a replica group. When creating a replica group, the system will automatically generate a "groupID" for a replica group.

name (string)

Replica group name. All replica group names are unique to each other in a database instance.

Format

The method "createRG()" contains the parameter "name", which is in the type of string. It should not be null.

(<"replica group name">)

Note:



- The value of the "name" cannot be a null string. It should not contain "." or "\$". The length of it should not be greater than 127B.

### Examples

Create a replica group named "group".

```
db.createRG("group")
```

## db.createUsr()

### NAME

createUsr - Create a database user with a user name and a password.

### SYNOPSIS

```
db.createUsr(<name>,<password>)
```

## CATEGORY

Sequoiadb

## DESCRIPTION

Create a database user with a user name and a password.

name (string)

User name.

password (string)

Password.

Format

The method "createUsr()" contains 2 parameters: name and password. Both of them are in the type of string. And they can be null string.

("<user name>","<password>")

Note:



- If there are any user accounts have been configured into the database, only the user with correct user name and password can log in and manipulate the database.

```
db = new Sdb("<hostname>", "<port>", "<name>", "<password>")
```

## Examples

Create a user with the user name "root" and the password "admin":

```
db.createUsr("root", "admin")
```

## db.dropCS()

## NAME

dropCS - Delete a specified collection space in a database instance.

## SYNOPSIS

```
db.dropCS(<name>)
```

## CATEGORY

Sequoiadb

## DESCRIPTION

Delete a specified collection space in a database instance.

name (string)

Collection space name. All the collection sapcenames in a database instance are unique to each other.

Format

The parameter "name" should be specified in the method "dropCS()". The type of collection space name should be string. It should be ensured that the collection space name exists in the database instance, or operation exception will occur.

("<Collection space name>")

**Note:**

- The value of "name" should not be a null string. It should not contain "." or "\$". And the length of it should not be greater than 127B. If the collection space name is invalid , the operation will fail.
- It should be ensured that the collection space name exists, or exception will occur.

**Examples**

Supposing that a collection space called "foo" exists, the following command will delete it.

```
db.dropCS("foo")
```

**db.dropDomain()****NAME**

dropDomain - Delete the specified domain.

**SYNOPSIS**

```
db.dropDomain(<name>)
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

Delete the specified domain.

name (string)

Domain name.

Format

The parameter "name" should be specified in the dropDomain() method, and value of "name" should exist in the system, otherwise an exception will occur.

```
{"name": "<domain name>"}
```

**Note:**

- Make sure that there is no data in the domain before deleting it.
- SYSDOMAIN cannot be deleted.

**Examples**

Delete an existent domain.

```
db.dropDomain('mydomain')
```

Delete a domain contains a collection space, return an exception.

```
>db.dropDomain('hello')
(nofile):0 uncaught exception: -256
Takes 0.1865s.
>getErr(-256)
Domain is not empty
```

## db.dropUsr()

### NAME

dropUsr - Delete an existent user and the corresponding password in the database.

### SYNOPSIS

```
db.dropUsr(<name>,<password>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Delete an existent user and the corresponding password in the database.

name (string)

User name.

password (string)

Password.

Format

The method "dropUsr()" contains 2 parameter: "name" and "password". Both of them are in the type of string.

("<user name>","<password>")

### Examples

Delete a user through the user name "root" and the password "admin".

```
db.dropUsr("root","admin")
```

## db.eval()

### NAME

eval - Fill in needed javaScript statements.

### SYNOPSIS

```
db.eval(<code>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION


Fill in needed javaScript statements. And it can also invoke defined procedures.

code (string)

JavaScript statements or defined procedure functions.

- Return the results according to the statements If success. And the returned value can be directly assigned to anther variable. For example: var a = db.eval(' db.foo.bar'); a.find();

- Return the error code and error information if fail.: { "errmsg": "(nofile):1 ReferenceError: sum is not defined","retCode": -152 }
- Nothing will be returned until the function is completely finished. Exit during run time will terminate the entire execution, and code has been executed will not be rolled back.
- There is a limit on the length of returned value, for more information, please check "Maximum length of inserted record in SequoiaDB" section.
- Support the definition of temporary functions, for example: db.eval(' function sum(x,y){return x+y;} sum(1,2)')
- The uses of global database are in the same way as createProcedure().

 Note: The use of any print statements in the statements are not recommended even though all the outputs will be blocked.

## Examples

Call a stored procedure function sum() in the eval() method.

```
//sum() method does not exist at the time of initialization, return a exception information.
> var a = db.eval(' sum(1,2) ');
{ "errmsg": "(nofile):1 ReferenceError: getCL is not defined", "retCode": -152 }
(nofile):0 uncaught exception: -152
//Initialization sum()
>db.createProcedure(function sum(x,y) {return x+y;})
//Call the sum()
>db.eval(' sum(1, 2) ')
3
```

Fill in javascript statement in the eval() method

```
>var rc = db.eval("db.foo.bar")
>rc.find()
{
  "_id": {
    "$oid": "5248d3867159ae144a000000"
  },
  "a": 1
}
{
  "_id": {
    "$oid": "5248d3897159ae144a000001"
  },
  "a": 2
}...
```

## db.exec()

NAME

exec - Execute SQL select statements.

SYNOPSIS

```
db.exec(<select sql>)
```

CATEGORY

Sequoiadb



**DESCRIPTION**

Execute SQL select statements.

**Examples**

Get all the records with "age = 20" from the collection named "my.my".

```
db.exec("select * from my.my where age = 20")
```

**db.execUpdate()****NAME**

execUpdate - Execute other SQL statements.

**SYNOPSIS**

```
db.execUpdate(<other sql>)
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

Execute other SQL statements.

**Examples**

Insert a new record into the collection named "my.my".

```
db.execUpdate("insert into my.my (name, age) values (' zhangshang', 30) ")
```

**db.flushConfigure()****NAME**

flushConfigure - Flush the configurations to the configuration files.

**SYNOPSIS**

```
db.flushConfigure(<rule>)
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

Flush the configurations to the configuration files.

rule (json object)

Flush rules.

Global

True indicates to flush the entire system's configurations. False indicates to flush the current node's configurations.

Global:true

### Examples

Flush database's configurations.

```
db.flushConfigure ({Global: true})
```

## db.getCS()

### NAME

getCS - Return the reference of a specified collection space.

### SYNOPSIS

```
db.getCS(<name>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Return the reference of a specified collection space.

name (string)

Collection space name. Collection space names are unique to each other in a database instance.

Format

getCS() only has one parameter name whose value should in a type of string.

("<collectionspace name>")



### Note:

- The value of param name can't be null, contain dot(.) or '\$', the length of it must be no more than 127B.
- The collectionspace must exist in the database.

### Examples

Assume a collection space named "foo" exists, It will return the reference of it.

```
db.getCS ("foo")
```

## db.getDomain()

### NAME

getDomain - Gets the specified domain.

### SYNOPSIS

```
db.getDomain(<name>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Gets the specified domain.

name (string)

Domain name.

Format

The parameter "name" should be specified in the getDomain() method, and value of "name" should exist in the collection space, otherwise an exception will occur.

```
{"name":"<domain name>"}
```



Note:

- SYSDOMAIN cannot be gotten.

## Examples

Get an existent domain.

```
var domain = db.getDomain('mydomain')
```

## db.getRG()

NAME

getRG - Return the reference of a replica group.

SYNOPSIS

```
db.getRG(<name> | <id>)
```

CATEGORY

Sequoiadb

DESCRIPTION

Return the reference of a replica group.

name (string)

Replica group name. Replica group names are unique to each other in a database instance.

id (int)

Replica group id. The system will automatically generate a replica group id.

Format

getRG() has two parameters "name" and "id". The value of "name" should be a string, the value of "id" is a int. If the replica group specified by "name" or "id" does not exist, an exception will occur.

```
("<replica group name>" | <id>)
```



Note:

- The value of "name" should not contain null string, "." or "\$". The length of it should not be greater than 127B.

## Examples

Specify the value of the "name", the command will return the corresponding reference of the replica group named "rg1".

```
db.getRG("rg1")
```

Specify the value of the "id", the command will return the corresponding reference of the replica group named "rg1".

```
db.getRG("1000")
```

## db.list()

NAME

list - Enumerate list.

SYNOPSIS

```
db.list(<listType>,[cond],[sel],[sort])
```

CATEGORY

Sequoiadb

DESCRIPTION

Enumerate list. List is a lightweight command to get the current system status.

listType (Enumeration)

listType, Please refer to List.

cond (json object)

Selecting condition. It just returns the records that match conditions in "cond". When it is null, return all records.

sel (json object)

Select fields to be returned. When it is null, return all the fields.

sort (json object)

Sort records. "1" means ascending order. "-1" means descending order.

Format

The method "list()" contains 4 parameters: listType, cond, sel and sort. The parameter "listType" is in the type of enumeration. Other parameters are in the type of json object. The format is as follows:

```
{"listType": "<List type>","cond": {"field name 1": {"operator 1": "value 1"}, "field name 2": {"operator 2": "value 2"}, ...}, "sel": {"field name 1": "", "field name 2": "", ...}, "sort": {"field name 1": -1 | 1, "field name 2": -1 | 1, ...}}
```



Note:

- For more details on the value of "listType", please refer to List .
- The value of "sel" is in the type of json object. The value of field is often specified as a null string. If the parameters are specified in this format: "{"field name 1": "value 1", "field name 2": "value 2", ...}", for the fields that do exist in the records, the value of these fields is actually useless; for the fields that do not exist in the records, "{"field name 1": "value 1", "field name 2": "value 2", ...}" will be returned.
- If a value of a field is in the type of array, users can use "." to invoke it and add double quotation.

Examples

Specify the value of "listType" as SDB\_LIST\_CONTEXTS:

```
db.list(SDB_LIST_CONTEXTS)
```

Return:

```
{
  "SessionID": 4,
  "Contexts": [ 0 ]
} ...
```

Specify the value of "listType" as SDB\_LIST\_STORAGEUNITS:

```
db.list(SDB_LIST_STORAGEUNITS)
```

Return:

```
{
  "Name": "foo",
  "ID": 4094,
  "Logical ID": 1,
  "PageSize": 4096,
  "Sequence": 1,
  "NumCollections": 1,
  "CollectionHWM": 3,
  "Size": 172032000
}
```

```
db.list(SDB_LIST_STORAGEUNITS, {"Logical ID": {$gt: 1}}, {Name: "space", ID: 2}, {Name: 1})
```

Return the records that contain the value of "Logical ID" greater than 1. Each record just returns two fields: "Name" and "ID". The values of "Name" in these records are sorted in ascending order.

```
{
  "ID": 4091,
  "Name": "foo"
}
{
  "ID": 4093,
  "Name": "name"
}...
```

## db.listBackup()

NAME

listBackup - View database backup.

SYNOPSIS

```
db.listBackup([options],[cond],[sel])
```

CATEGORY

Sequoiadb

DESCRIPTION

View database backup.

options (json object)

Specify the name of the backup, the replica group, and number of copies, etc.

GroupID

Specify the ID of the backup replica group to output. List all replica groups in default.

GroupID:1000 or GroupID:[1000, 1001]

GroupName

Specify the name of the backup replica group to output. List all the replica groups in default.

GroupName:"data1" or GroupName:["data1", "data2"]

Name

Backup name, list all the backups in default.

Name:"backup-2014-1-1"

Path

Backup path, the default path is the one specified in the configuration. The path supports wildcards(%g%G: group name, %h%H:host name, %s%S:service name).

Path:"/opt/sequoiadb/backup/%g"

IsSubDir

Whether the path configured in the above Path parameter is a subdirectory of the backup path in the configuration. False in default.

IsSubDir:false

Prefix

Backup prefix, supports wildcards(%g,%G,%h,%H,%s,%S), default is empty.

Prefix:"%g\_bk\_"

cond (json object)

Filter conditions.

sel (json object)

Output fields.

Examples

View all database backup information.

```
db.listBackup()
```

## db.listCollections()

NAME

listCollections - Enumerate collections.

SYNOPSIS

```
db.listCollections()
```

CATEGORY

Sequoiadb

DESCRIPTION

Enumerate collections. It will return the information of every collection in a collection space.

Examples

```
db.listCollections()
```

Return :

```
{
```

```

    "Name": "foo.bar",
    "Details": [
      {
        "ID": 0,
        "Logical ID": 1,
        "Sequence": 1,
        "Indexes": 2,
        "Status": "Normal"
      }
    ]
  }
}

```

## db.listCollectionSpaces()

NAME

listCollectionSpaces - The collection spaces in the database.

SYNOPSIS

```
db.listCollectionSpaces()
```

CATEGORY

Sequoiadb

DESCRIPTION

The collection spaces in the database.

Examples

```
db.listCollectionSpace()
```

Returns:

```

{
  "Name": "foo"
}...

```

## db.listDomains()

NAME

listDomains - Emuratedoamins.

SYNOPSIS

```
db.listDomains()
```

CATEGORY

Sequoiadb

DESCRIPTION

Emuratedoamins. This method will list every domain created by the users in the current system.

### Examples

`db.listDomains()`

Returns:

```
{
  "_id": {
    "$oid": "539ea19669d195f36432111a"
  },
  "Name": "hello",
  "Groups": [
    {
      "GroupName": "data1",
      "GroupID": 1000
    },
    {
      "GroupName": "data2",
      "GroupID": 1001
    }
  ]
}
```

### `db.listProcedures()`

NAME

`listProcedures` - Enumerate the stored procedure functions.

SYNOPSIS

`db.listProcedures([cond])`

CATEGORY

Sequoiadb

DESCRIPTION

Enumerate the stored procedure functions.

`cond` (json)

If `cond` is "", enumerate all the functions. Otherwise, enumerate all the functions satisfy the criteria.

`listProcedures()` has only one parameter 'cond' which is a json object. It returns the all the functions if there is no given criteria, otherwise all the functions satisfy the criteria will be returned.

### Examples

Return the information of all the functions.

```
>db.listProcedures ()
{ "_id" : { "$oid" : "52480389f5ce8d5817c4c353" }, "name" : "sum", "func" : "function sum(x,
y) {
  return x + y;
}", "funcType" : 0 }
{ "_id" : { "$oid" : "52480d3ef5ce8d5817c4c354" }, "name" : "getAll", "func" : "function
getAll() {
  return db.foo.bar.find();
}", "funcType" : 0 }
...
```



Returns the information of the function named "sum".

```
>db.listProcedures ({name: "sum"})
{ "_id" : { "$oid" : "52480389f5ce8d5817c4c353" }, "name" : "sum", "func" : "function sum(x,
y) {
  return x + y;
}", "funcType" : 0 }
```

## db.listReplicaGroups()

NAME

listReplicaGroups - List replica groups.

SYNOPSIS

```
db.listReplicaGroups()
```

CATEGORY

Sequoiadb

DESCRIPTION

List replica groups.

Examples

Return the information of every replica group:

```
> db.listReplicaGroups ()
Return: {
  "Group": [
    {
      "dbpath": "/opt/sequoiadb/data/11800",
      "HostName": "vmsvr2-suse-x64",
      "Service": [
        {
          "Type": 0,
          "Name": "11800"
        },
        {
          "Type": 1,
          "Name": "11801"
        },
        {
          "Type": 2,
          "Name": "11802"
        },
        {
          "Type": 3,
          "Name": "11803"
        }
      ]
    },
    "NodeID": 1000
  ],
  {
    "dbpath": "/opt/sequoiadb/data/11850",
    "HostName": "vmsvr2-suse-x64",
    "Service": [
      {
        "Type": 0,
        "Name": "11850"
      }
    ]
  }
}
```

```

    },
    {
      "Type": 1,
      "Name": "11851"
    },
    {
      "Type": 2,
      "Name": "11852"
    },
    {
      "Type": 3,
      "Name": "11853"
    }
  ],
  "NodeID": 1001
}
],
"GroupID": 1001,
"GroupName": "group",
"PrimaryNode": 1001,
"Role": 0,
"Status": 1,
"Version": 5,
"_id": {
  "$oid": "517b2fc33d7e6f820fc0eb57"
}
}

```

This replica group has 2 nodes: 11800 and 11850. Node 11850 is the master node.

## db.listTasks()

NAME

listTasks - View the background tasks of the database.

SYNOPSIS

`db.listTasks([cond],[sel],[orderBy],[hint])`

CATEGORY

Sequoiadb

DESCRIPTION

View the background tasks of the database.

cond (json object)

Tasks filter.

sel (json object)

Tasks selecting condition.

hint (json object)

Reservations.

Examples

List all the background tasks in the system.

```
db.listTasks()
```

## db.removeBackup()

### NAME

removeBackup - Delete the database backup.

### SYNOPSIS

```
db.removeBackup([options])
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Delete the database backup.

options (json object)

Set the backup name, specify the replica group, the backup path and other parameters.

GroupID

Specify the ID of the backup replica group needs to be deleted. Delete all replica groups in default.

GroupID:1000 or GroupID:[1000, 1001]

GroupName

Specify the name of the backup replica group needs to be deleted. Delete all replica groups in default.

GroupName:"data1" or GroupName:["data1", "data2"]

Name

Backup name, delete all the backups in default.

Name:"backup-2014-1-1"

Path

Backup path, the default path is the one specified in the configuration. The path supports wildcards(%g%G: group name, %h%H:host name, %s%S:service name).

Path:"/opt/sequoiadb/backup/%g"

IsSubDir

Whether the path configured in the above Path parameter is a subdirectory of the backup path in the configuration. False in default.

IsSubDir:false

Prefix

Backup prefix, supports wildcards(%g,%G,%h,%H,%s,%H), default is empty.

Prefix:"%g\_bk\_"

### Examples

Delete the backup named "backup-2014-1-1" from the database.

```
db.removeBackup({Name: "backup-2014-1-1"})
```

## db.removeProcedure()

### NAME

removeProcedure - Deletes the specified function name.

### SYNOPSIS

```
db.removeProcedure(<function name>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Deletes the specified function name, The specified function name must exist, otherwise exceptions will occur.

function name (string)

function name.

removeProcedure() method has one parameter 'function name' in a string type. Its value should already exist, otherwise exceptions will occur.

### Examples

Delete the sum function:

```
db.removeProcedure("sum")
```

Make sure that the function name specified in the removeProcedure() method be exactly the same as the name in its definition. Otherwise, it will return a failure. db.removeProcedure('sum') call will return a failure.

## db.removeRG()

### NAME

removeRG - Delete the specified replica group in the current database.

### SYNOPSIS

```
db.removeRG(<name>)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Delete the specified replica group in the current database.

name (string)

Replica group name, All the replica group names are unique to each other in a database instance.

Format

removeRG() only has one "name" parameter, it is required and its value should be a string.

(<"replica group name">)



Note:

- Replica group name must exist.

### Examples

Delete the replica group named "group".

```
db.removeRG("group")
```

## db.resetSnapshot()

NAME

resetSnapshot - Reset snapshot.

SYNOPSIS

```
resetSnapshot([cond])
```

CATEGORY

Sequoiadb

DESCRIPTION

Reset snapshot.

cond (json object)

Selecting condition. It merely resets snapshot records that match the conditions in "cond". If it is null, it will reset all the snapshot records.

Format

The method "resetSnapshot()" contains the parameter "cond". It is a json object.

```
{["cond":{"Field name 1":{"match 1":"value 1"},"Field name 2":{"match 2":"value 2"}...}]}
```

### Examples

Reset snapshots that contain the value of "SessionID" greater than 1.

```
db.resetSnapshot({SessionID: {$gt:1}})
```

## db.setSessionAttr()

NAME

setSessionAttr - Set session attributes.

SYNOPSIS

```
db.setSessionAttr(<options>)
```

CATEGORY

Sequoiadb

**DESCRIPTION**

Set session attributes.

options (json object)

Session attributes' options.

PreferedInstance

A preferred database instance mark for sessions' read operations. Its values contain:"m"/"M"/"s"/"S"/"a"/"A"/1-7 which stands for master/slave/anyone/node1-node7 respectively.

PreferedInstance:"M"

**Examples**

Set the preferred instance as master so that the session will preferentially read the data from a master database instance.

```
db.setSessionAttr({PreferedInstance: "M"})
```

**db.snapshot()****NAME**

snapshot - It lists snapshots.

**SYNOPSIS**

```
snapshot(<snapType>,[cond],[sel],[sort])
```

**CATEGORY**

Sequoiadb

**DESCRIPTION**

It lists snapshots. Snapshot() is a command used to get the current status of system.

snapType (list)

Snapshot type.Please refer to Snapshot.

cond (json object)

Selecting condition. Only return the snapshot information of the nodes and replica groups that match the selecting conditions specified in the 'cond' field. When the field is null, return the cluster's information.

sel (json object)

Selects the returned field. Returns all the fields when it is null.

sort (json object)

Sorts the returned records according to the selected fields. 1 for ascending order, -1 for descending order.

Format

Snapshot() has four parameters: snapType, cond, sel, sort. The type of "snapType" is enumeration. Other fields are json objects. The format is as follow:

```
{"snapType": "<snapshot type>","cond": {"Field name 1": {"operation 1": "value 1"}, "Field name 2": {"Operation 2": "value 2"}...}}
```



Note:

- For more details on the value of snapType, please refer to Snapshot.
- The parameter "sel" is a json object. The value of field is often specified as a null string. If the parameters are specified in this format: "{"field name 1": "value 1", "field name 2": "value 2", ...}", for the fields that do exist in the records, the value of these fields is actually useless; for the fields that do not exist in the records, "{"field name 1": "value 1", "field name 2": "value 2", ...}" will be returned.
- If the value of a field is in the type of array. Elements are invoked by "."(dot) and double quotations.

## Examples

Set the value of "snapType" as SDB\_SNAP\_CONTEXTS:

```
db. snapshot (SDB_SNAP_CONTEXTS)
```

Returns context snapshot of the entire cluster:

```
{
  "SessionID": "vmsvr1-cent-x64-1:11820:22",
  "Contexts": [
    {
      "ContextID": 8,
      "Type": "DUMP",
      "Description": "BufferSize: 0",
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-12-28-16.07.59.146399"
    }
  ]
}
{
  "SessionID": "vmsvr1-cent-x64-1:11830:22",
  "Contexts": [
    {
      "ContextID": 6,
      "Type": "DUMP",
      "Description": "BufferSize: 0",
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-12-28-16.07.59.147576"
    }
  ]
}
{
  "SessionID": "vmsvr1-cent-x64-1:11840:23",
  "Contexts": [
    {
      "ContextID": 7,
      "Type": "DUMP",
      "Description": "BufferSize: 0",
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-12-28-16.07.59.148603"
    }
  ]
}
```

Get the snapshot of the specified replica group by group name or group ID, for example:

```
db. snapshot (SDB_SNAP_CONTEXTS, {GroupName: 'data1'})
```

Return the snapshot of a group named "data1".

```
db.snapshot (SDB_SNAP_CONTEXTS, {GroupID: 1000})
```

Return the snapshot of a replica group with a group ID of 1000.

```
{
  "SessionID": "vmsvr1-cent-x64-1:11820:22",
  "Contexts": [
    {
      "ContextID": 11,
      "Type": "DUMP",
      "Description": "BufferSize: 0",
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-12-28-16.13.57.864245"
    }
  ]
}
{
  "SessionID": "vmsvr1-cent-x64-1:11840:23",
  "Contexts": [
    {
      "ContextID": 10,
      "Type": "DUMP",
      "Description": "BufferSize: 0",
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-12-28-16.13.57.865103"
    }
  ]
}
```

Get the snapshot of a node by specifying "group name + host name + service name" or "group ID + node ID", for example:

```
db.snapshot (SDB_SNAP_CONTEXTS, {GroupName: 'data1', HostName: "vmsvr1-cent-x64-1", svcname: "11820"})
```

```
db.snapshot (SDB_SNAP_CONTEXTS, {GroupID: 1000, NodeID: 1001})
```

```
{
  "SessionID": "vmsvr1-cent-x64-1:11820:22",
  "Contexts": [
    {
      "ContextID": 11,
      "Type": "DUMP",
      "Description": "BufferSize: 0",
      "DataRead": 0,
      "IndexRead": 0,
      "QueryTimeSpent": 0,
      "StartTimestamp": "2013-12-28-16.13.57.864245"
    }
  ]
}
```

Get the snapshot of a node by specifying "host name + service name", for example:

```
db.snapshot (SDB_SNAP_CONTEXTS, {HostName: "vmsvr1-cent-x64-1", svcname: "11820"})
```



## db.startRG()

NAME

startRG - Start a specified replica group.

SYNOPSIS

```
db.startRG(<name1>,[name2],...)
```

CATEGORY

Sequoiadb

DESCRIPTION

Start a specified replica group. A node can only be created after a replica group is already started. This method is equivalent to rg.start().

name (string)

The name of a replica group.

Format

The method "db.startRG()" contains a parameter called "name". The type of it is string. It is the name of the replica group which is about to be started.

("<replica group name>")

Note:



- Exception will occur when the specified replica group doesn't exist or it has been started.

Examples

The following command starts a replica group called "group":

```
db.startRG("group")
```

## db.transBegin()

NAME

transBegin - Open the transaction.

SYNOPSIS

```
db.transBegin()
```

CATEGORY

Sequoiadb

DESCRIPTION

Open the transaction function. Transaction in SequoiaDB is a series of operations performed as a single logical unit. Transaction processing can ensure that all the non-transactional operations are successfully completed in the unit, otherwise it would not permanently update the data-oriented resources.

### Examples

Open the transaction commands:

```
db.transBegin()
```

## db.transCommit()

### NAME

transCommit - Transaction commits.

### SYNOPSIS

```
db.transCommit()
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Commit transactions. When transaction function is on, if an operation of a single logical unit performed without any exception, execute the transCommit() command, the data in the database will be updated.

### Examples

Transaction commit command:

```
db.transCommit()
```

## db.transRollback()

### NAME

transRollback - Transaction rollback.

### SYNOPSIS

```
db.transRollback()
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Roll back the transactions. When transaction function is on, if any exceptions occurred when performing operations in a single logical unit, execute the transRollback() command, the database will back to its original state.

### Examples

Transaction rollback command:

```
db.transRollback()
```

## db.waitTasks()

### NAME

waitTasks - Wait for the end or cancellation of specified task synchronously.

### SYNOPSIS

```
db.waitTasks(<id1>,[id2],...)
```

### CATEGORY

Sequoiadb

### DESCRIPTION

Wait for the end or cancellation of specified task synchronously.

id1,id2,... (Integer)

Task ID.

### Examples

Synchronously wait the data split task to finish.

```
var taskid1 = db.test.test.splitAsync("db1", "db2", 50);
var taskid2 = db.my.my.splitAsync("db3", "db4", 50) ;
db.waitTasks( taskid1, taskid2 )
```

## db.collectionspace.createCL()

### NAME

createCL - create a new collection

### SYNOPSIS

```
db.collectionspace.createCL(<name>,[option])
```

### CATEGORY

Collectionspace

### DESCRIPTION

Create a collection in a specified collection space.

Collection is a logical object which stores records. Each record should belong to one and only one collection.

name (string)

The name of the collection, should be unique to each other in a collectionspace.

options (object)

The parameters of the collection, could be a combination of the following options

ShardingKey (object)

The sharding key of the collection.

ShardingType (string)

The sharding type of the collection, could be one of the following

"hash"

Sharding by hash

"range"

Sharding by range

Partition (number)

Specify the number of slices, only valid for { ShardingType:"hash" }

ReplSize (number)

Number of WriteConcern. Minimum 0 and maximum 7.

0

WriteConcern equal to the number of nodes in each replica group

1 - 7

WriteConcern = 1-7

Compressed (boolean)

ompress the collection or not

IsMainCL (boolean)

Partitioned collection

AutoSplit (boolean)

automatic split.

Group (string)

Specify a replica group which the collection should be in.

Group:<group name>



Note:

- The parameter "ShardingKey" is a JSON object. Each of the fields in the JSON object corresponds to that in shard key. The value is 1 or -1, representing ascending and descending order.
- The parameter "ReplSize" is a JSON object, the value of it is a int type, Be used to set the number of writed data nodes, the value can not be greater than the number of nodes in the current data group.
- The parameter "Compressed" is a JSON object, the value of it is boolean type, if the value is "true", then standing for the data of the collection will be stored in compressed form , otherwise, data will be stored in normal .
- when the options have one or more params, please use comma(,) to separate.
- The value of "name" should not be null string, "." or "\$". The length of it should not be greater than 127B, or exception will occur.
- AutoSplit must cooperate with hash partitioning and domain. AutoSplit and Group cannot be set at the same time.
- AutoSplit and Group cannot be set at the same time.
- If AutoSplit is not specOKified in a collection, the value of AutoSplit will come from domain.
- Group must exist in a domain, which contains a collection space, which has the group(All replica group belong to SYSDOMAIN, that is to say, a collection space can be distributed into any replica group if no specific group is given)

## RETURN VALUE

On success, createCL() creates a new collection and return the object. On error, exception will be thrown.

## ERRORS

SDB_OOM (-2)	Unable to allocate memory for creating collection
SDB_NOSPC (-11)	Out of space
SDB_DMS_EXIST (-22)	Collection already exist
SDB_DMS_CS_NOTEXIST (-34)	Collection space does not exist

## Examples

Create collection "bar" in collection space "foo" without shard key. If the collection space "foo" doesn't exist, it will automatically generate collection space "foo".

```
db.foo.createCL("foo")
```

Create a compressed normal collection "bar" in collection space "foo" with the hash shard key "age" and in ascending order, using 65535 partitions with writeconcern 1.

```
db.foo.createCL("bar", {ShardingKey: {age: 1}, ShardingType: "hash", Partition: 65535, ReplSize: 1, Compressed: true, IsMainCL: false})
```

**db.collectionspace.dropCL()**

## NAME

dropCL - Delete a specified collection in a specified collection space.

## SYNOPSIS

```
db.collectionspace.dropCL(<name>)
```

## CATEGORY

Collectionspace

## DESCRIPTION

Delete a specified collection in a specified collection space.

name (string)

Collection name. Collection names should be unique to each other in a collection space.

Format

The parameter "name" should be specified in the method "dropCL()". It should be ensured that the collection name exists in the collection space, or operation exception will occur.

```
{"name": "<collection name>"}
```



## Note:

- The value of "name" should not be null string. It should not contain "." or "\$". And the length of it should not be greater than 127B. If the collection name is invalid, the operation will fail.
- It should be ensured that the collection name exists in a collection space, or exception will occur.

### Examples

Supposing that the collection "bar" exists in the collection space "foo", the following command will delete it.

```
db.foo.dropCL("bar")
```

## db.collectionspace.getCL()

### NAME

getCL - Returns the reference of collection in a collection space.

### SYNOPSIS

```
db.collectionspace.getCL(<name>)
```

### CATEGORY

Collectionspace

### DESCRIPTION

Returns the reference of collection in a collection space.

name (string)

Collection name. In a collection space, collection name should be unique to each other.

Format

The parameter "name" in the method "getCL()" should be specified and the "name" should be valid, or exception will occur.

```
{"name": "<collection name>"}
```



### Note:

- The value of "name" should not be null string. It should not contain "." or "\$". The length of should not be greater than 127B. If the value of "name" is invalid, exception will occur.
- Make sure that the collection name exists in the collection space, or operation exception will occur.

### Examples

Supposing that the collection name "bar", the command returns the reference of the collection "bar" in the collection space "foo".

```
db.foo.getCL("bar")
```

## db.collectionspace.collection.aggregate()

### NAME

aggregate - retrieve the documents from the collection in the SequoiaDB and return a cursor.

### SYNOPSIS

```
db.collectionspace.collection.aggregate(<subOp>...)
```

### CATEGORY

Collection

## DESCRIPTION

The functions of `aggregate()` and `find()` are pretty similar to each other. It also retrieves the documents from the collection in the SequoiaDB and return a cursor.

`subOp` (json object)

Sub-operation, 1 to N sub-operations can be filled in the `aggregate()` method.

`aggregate()` method has only one parameter `subOp`, which represents 1~N sub-operations, each sub-operation is a json object, separated by commas sub-operation. Aggregation framework supports the following Sub-operation parameters:

`$project`

Select the output fields by specifying field names, 1 stands for output, 0 means do not output. The rename of fields can be realized.

```
{ $project: { field1: 1, field: 0, aliase: "$field3" } }
```

`$match`

Select records which match the criteria from the collection, equivalent to the "WHERE" command in SQL.

```
{ $match: { field: { $lte: value } } }
```

`$limit`

Limit the number of returned records.

```
{ $limit: 10 }
```

`$skip`

Control the starting point of the result set, in other words, skip specified number of records in the result set.

```
{ $skip: 5 }
```

`$group`

Divide the records into groups, similar to the "group by" command in SQL, specify the grouping field by "\_id".

```
{ $group: { _id: "$field" } }
```

`$sort`

Sort the result set. 1 for ascending while 0 for descending.

```
{ $sort: { field1: 1, field2: -1, ... } }
```

Note:



- Description: `aggregate()` can have any number of sub-operations, but please be careful about the syntax of the parameters.

## Examples

Assume a collection stores records in following format:

```
{
  no: 1000,
  score: 80,
  interest: ["basketball", "football"],
  major: "Computer Science and Technology",
  dep: "Computer Academy",
  info: {
    name: "Tom",
    age: 25,
    gender: "male"
  }
}
```

```
}
```

Select records according to the criteria, and specify the returned field name.

```
db.collectionspace.collection.aggregate({$match: {$and: [{no: {$gt: 1002}}, {no: {$lt: 1015}}],
{dep: "Computer Academy"}]}},
{$project: {no: 1, "info.name": 1, major: 1}})
```

This aggregate operation firstly uses `$match` to select the records match the criteria, then uses `$project` to specify the names of the returned fields.

```
{
  "no": 1003,
  "info.name": "Sam",
  "major": "Computer Software and Theory"
}
{
  "no": 1004,
  "info.name": "Coll",
  "major": "Computer Engineering"
}
{
  "no": 1005,
  "info.name": "Jim",
  "major": "Computer Engineering"
}
```

Select records by criteria and divide the selected records into groups.

```
db.collectionspace.collection.aggregate({$match: {dep: "Computer Academy"}}, {$group:
{_id: "$major", Major: {$first: "$major"},
avg_age: {$avg: "$info.age"}}})
```

This operation firstly uses `$match` to select records that match the selecting criteria, then uses `$group` to divide the selected records by field "major", and uses `$avg` to return the average of the "age" field in each group.

```
{
  "Major": "Computer Engineering",
  "avg_age": 25
}
{
  "Major": "Computer Science and Technology",
  "avg_age": 22.5
}
{
  "Major": "Computer Software and Theory",
  "avg_age": 26
}
```

Select records by criteria, then group and sort the selected records, limit the starting point of the result set and the number of returned records.

```
db.collectionspace.collection.aggregate({$match: {interest: {$exists: 1}}}, {$group:
{_id: "$major", avg_age: {$avg: "$info.age"},
major: {$first: "$major"}}}, {$sort: {avg_age: -1, major: -1}}, {$skip: 2}, {$limit: 3})
```

This aggregate operation firstly uses `$match` to select records match the criteria, then uses `$group` to group the records by "major", uses `$avg` to return the average of "age" field in each group, and sort the records in descending order, uses `$skip` to specify the starting point of the returned result, and `$limit` to limit the number of returned records.

```
{
  "avg_age": 25,
  "major": "Computer Science and Technology"
}
{
```



```

    "avg_age": 22,
    "major": "Computer Software and Theory"
  }
  {
    "avg_age": 22,
    "major": "Physics"
  }

```

## db.collectionspace.collection.alter()

### NAME

alter - Modify the collection properties.

### SYNOPSIS

```
db.collectionspace.collection.alter(<options>)
```

### CATEGORY

Collection

### DESCRIPTION

Modify the collection properties.

options (Json Object)

Attributes to be modified.

ReplSize

Number of replica.

ReplSize:<int32>

ShardingKey

Key to shard.

ShardingKey:{<field 1>:<1 | -1>,[<field 2>:<1 | -1>, ...]}

ShardingType

Sharding type, the default value is range.

ShardingType:"hash"|"range"

Partition

Number of partitions. Fill out when there is a hash partitioning. It represents the number of hash partitioning. Its value must be a power of 2, ranging from 2<sup>3</sup> to 2<sup>20</sup>.

Partition:<number of partitioning>



#### Note:

- Usage of ShardingKey, ShardingType, Partition, please check db.collectionspace.createCL().
- Partitioning collection cannot alter attributes related to partitioning.
- After altering to partitioning collection, we must split manually.

### Examples

Create a collection.

```
db.foo.createCL('bar')
```

Alter to partitioning collection.

```
db.foo.bar.alter({ShardingKey: {a: 1}, ShardingType: "hash"})
```

## db.collectionspace.collection.attachCL()

### NAME

attachCL - Attach child partition-collections to the main partition.

### SYNOPSIS

```
db.collectionspace.collection.attachCL(<subCLFullName>,<options>)
```

### CATEGORY

Collection

### DESCRIPTION

Attach child partition-collections to the main partition.

subCLFullName (string)

The child partition-collection's full name(include the name of collection space).

options (json object)

Partition range, includes two fields named "LowBound" and "UpBoumd".

e.g., {LowBound:{a:0},UpBound:{a:100} stands for choosing the range of field a:[0, 100)

### Examples

Mount the child partition-collections to the specified main collection.

```
db.foo.year.attachCL("foo2. January", {LowBound: {date: "20130101"}, UpBound: {date: "20130131"}})
```

## db.collectionspace.collection.count()

### NAME

count - Return the total amount of records in a specified collection in a collection space.

### SYNOPSIS

```
db.collectionspace.collection.count([cond])
```

### CATEGORY

Collection

### DESCRIPTION

Return the total amount of records in a specified collection in a collection space.

cond (json object)

Selecting condition. If it is null, it will count all the records in the collection. If it is not null, it will count records that match the conditions.

Format

The method "count()" contains field named "cond". It is a JSON object.

```
{{"Field name 1":{"match 1":"value 1"},"Field name 2":{"match 2":"value 2"},...}}
```

## Examples

Count all the records in collection "bar" without specifying "cond".

```
db.foo.bar.count()
```

Count records that both contain the value "Tom" on the field "name" and the value on the field "age" is greater than 25.

```
db.foo.bar.count({name: "Tom", age: {$gt: 25}})
```

## db.collectionspace.collection.createIndex()

### NAME

createIndex - Create an index for the collection to accelerate query.

### SYNOPSIS

```
db.collectionspace.collection.createIndex(<name>,<indexDef>,[isUnique],[enforced])
```

### CATEGORY

Collection

### DESCRIPTION

Create an index for the collection to accelerate query.

name (string)

Index name. It should be unique in a collection.

indexDef (json object)

Index key. It contains one or more objects that specify index fields and order direction. "1" means ascending order. "-1" means descending order.

isUnique (Boolean)

It shows whether the index is unique. The default value is "false". When it is "true", the index is unique.

enforced (Boolean)

Optional, whether the index is mandatorily unique or not. Its default value is false, and it becomes effective when "isUnique" is true. When it is true, it means that in the premise of "isUnique" is true, no more than one null index key can exist in this index.

Format

The method "createIndex()" contains three parameters: "name", "indexDef" and "isUnique". The value of "name" should be in the type of string. The field "indexDef" is defined as a JSON object, which contains at least one field. The field name is index name. The value of it is 1 or -1. "1" means ascending order. "-1" means descending order. The parameter "isUnique" is in the type of boolean. Its default value is "false".

```
{"name":"<index name>","indexDef":{"<index field 1>":<1 | -1> [,"<index field 2>":<1 | -1>...]},"isUnique":<true | false>},"enforced":<true | false>}}
```



Note:

- There should not be any exactly same records in the fields that are specified by the unique index in a collection.
- Index name should not be null string. It should not contain "." or "\$". The length of it should be no more than 127B.

### Examples

Create an unique index named "ageIndex" on the field "age" in collection "bar". The records are in ascending order on the field "age".

```
db.foo.bar.createIndex("ageIndex", {age: 1}, true)
```

## db.collectionspace.collection.detachCL()

### NAME

detachCL - Detach the child partition-collections from the main partition-collection.

### SYNOPSIS

```
db.collectionspace.collection.detachCL(<subCLFullName>)
```

### CATEGORY

Collection

### DESCRIPTION

Detach the child partition-collections from the main partition-collection.

subCLFullName (string)

Child partition-collection's full name(includes the collection space name).

### Examples

Detach the child partition-collection from the specified main partition-collection.

```
db.foo.year.detachCL("foo2. January")
```

## db.collectionspace.collection.dropIndex()

### NAME

dropIndex - Delete specified index in a collection.

### SYNOPSIS

```
db.collectionspace.collection.dropIndex(<name>)
```

### CATEGORY

Collection

### DESCRIPTION

Delete specified index in a collection.

name (string)

Index name. Index names should be unique to each other in the same collection.

Format

Then method "dropIndex()" contains a field named "name". The value of "name" should be string.

```
{"name": "<index name>"}
```

**Note:**

- When deleting index, it should be ensured that the index name is in the collection.
- The index name can't be null, contain dot(.) or "\$", and the length of it must be no more than 127B.

**Examples**

Supposing that the index "ageIndex" exists in the collection "bar", the following command will delete it.

```
db.foo.bar.dropIndex("ageIndex")
```

**db.collectionspace.collection.find()****NAME**

find - Select specified records.

**SYNOPSIS**

```
db.collectionspace.collection.find([cond],[sel])
```

**CATEGORY**

Collection

**DESCRIPTION**

Select specified records. Return a cursor on the selected records. Cursor is a pointer in SequoiaDB, it points to a query result set, the search result can be iterated in a client.

cond (json object)

Selecting condition. If it is null, it will find all the records. If it is not null, it will find records that matches the condition.

sel (json object)

It chooses fields to be returned. If it is null, it will return all the records. If a field doesn't exist, it will return null string.

Format

The definition of "find()" contains 2 parameters: cond and sel, which are both in the type of JSON object. The parameter "cond" specifies the matching condition. The parameter "sel" specifies the fields to be returned.

```
{{"field name 1":{"match 1":"value 1","field name 2":{"match 2":"value 2"},...}},{"field name 1":"","field name 2":"","...}]}
```

**Note:**

- The parameter "sel" is in the type of a json object. The value of field is often specified as a null string. If the parameters are specified in this format: '{"field name 1": "value 1", "field name 2": "value 2",...}', for the fields that do exist in the records, the value of these fields is actually useless; for the fields that do not exist in the records, '{"field name 1": "value 1", "field name 2": "value 2",...}' will be returned.

**Examples**

Find all the records without specifying the field "cond" and "sel".

```
db.foo.bar.find()
```

Find records that matches the value of the parameter "cond".

```
db.foo.bar.find({age: {$gt: 25}, name: "Tom"})
```

This operation will return records that contain the value of "age" greater than 25 and the value "Tom" on the field "name".

Specify the fields to return by setting the content of the parameter "sel". For example, there are 2 records: "{age:25,type:"system"}" and "{age:20,name:"Tom",type:"normal"}".

```
db.foo.bar.find(null, {age: "", name: ""})
```

This operation return the value of "age" and "name". So it will return: "{age:25,name:""}, {age:20,name:"Tom"}". Although the first record doesn't contain the field, it will return "name:"".

## db.collectionspace.collection.getIndex()

NAME

getIndex - Return the reference of specified index.

SYNOPSIS

```
db.collectionspace.collection.getIndex(<name>)
```

CATEGORY

Collection

DESCRIPTION

Return the reference of specified index.

name (string)

Index name, all the index names in a collection should be unique to each other.

Format

The method "getIndex()" contains the field "name", which is in the type of string.

```
{"name": "<index name>"}
```



Note:

- It should be ensured that the index name exists in the collection before users get index reference.
- Index name should not contain null string, "." or "\$". The length of it should not be greater than 127B.

Examples

Supposing that the index "ageIndex" exists in collection "bar", the following command will return the reference of it.

```
db.foo.bar.getIndex("ageIndex")
```

## db.collectionspace.collection.insert()

NAME

insert - Insert records into a specified collection.

SYNOPSIS

```
db.collection.space.collection.insert(<doc|docs>,[flag])
```

## CATEGORY

Collection

## DESCRIPTION

Insert records into a specified collection. If the collection space or collection doesn't exist, please create a new collection space. For example, the command "db.createCS("foo")" creates a new collection space named "foo". Then create a new collection in collection space. For instance, the command "db.foo.createCL("bar")" creates a collection named "bar" under the collection space "foo" so that users can insert data into the collection "bar".

doc|docs (json object)

Document record. The parameter "doc" means one record. The parameter "docs" means more than one record.

flag (int)

SDB\_INSERT\_RETURN\_ID or SDB\_INSERT\_CONTONDUP. The former one is valid when insert a single record, the latter one is valid when insert multiple records, it means that when multiple records are inserted, if records with repeated "\_id" field occurs, these records will be skipped and the database will continue with subsequent records. In the default case, if records with repeated "\_id" field occurs, the insert action will be terminated and the subsequent records will not be inserted..

Format

The definition of "insert()" contains two fields: doc|docs and flag.

doc: {{"< field name 1>":"< value >","< field name 2>":"< value >","...}  
[SDB\_INSERT\_CONTONDUP|SDB\_INSERT\_RETURN\_ID]}

docs: ([ {"< field name 1>":"< value >","< field name 2>":"< value >","..."}, {"< field name 1>":"< value >","< field name 2>":"< value >","..."},... ])



Note:

- If the value of "\_id" is not specified in a record that is about to be inserted, the system will generate the value of "\_id" automatically to ensure the uniqueness of the record.

## Examples

Insert a record without the value of "\_id".

```
db.foo.bar.insert({name:"Tom",age:20})
```

This operation inserts a new record into collection "bar". In this record, the value of "name" is "Tom", and the value of "age" is 20. The value of "\_id" is uniquely generated:

```
{ "_id": { "$oid": "515152ba49af3952000000000" }, "name": "Tom", "age": 20 }
```

Insert a record that contains the value of "\_id".

```
db.foo.bar.insert({_id:10,age:20})
```

This operation inserts a new record into the collection "bar". The value of "\_id" is 10. The value of "age" is 20:

```
{ "_id": 10, "age": 20 }
```

Insert more than one record.

```
db.foo.bar.insert([ {_id:20,name:"Mike",age:15}, {name:"John",age:25,phone:123} ])
```

This operation will insert 2 records into the collection "bar":

1) In the first record, the value of "\_id" is 20. The value of "name" is "Mike". The value of "age" is 15.

2) In the second record, the value of "\_id" is generated by the system. The value of "name" is "John". The value of "age" is 25. The value of "phone" is 123.

```
{
  "_id": 20,
  "name": "Mike",
  "age": 15
}
```

```
{
  "_id": { "$oid": "5151557a49af395200000001" },
  "name": "John",
  "age": 25,
  "phone": 123
}
```

Insert records with repeated "\_id" field.

```
db.foo.bar.insert([ {_id: 1, a: 1 }, {_id: 1, b: 2 }, {_id: 3, c: 3} ], SDB_INSERT_CONTONDUP)
```

This operation will insert two records in collection "bar".

```
{
  "_id": 1,
  "a": 1,
}
{
  "_id": 3,
  "c": 3
}
```

## db.collectionspace.collection.listIndexes()

NAME

listIndexes - Enumerate indexes.

SYNOPSIS

```
db.collectionspace.collection.listIndexes()
```

CATEGORY

Collection

DESCRIPTION

Enumerate indexes. It will show information of all indexes in the specified collection.

Examples

The following command will return information of all the indexes in the collection "bar".

```
db.foo.bar.listIndexes()
```

## db.collectionspace.collection.remove()

NAME

remove - Delete records from the current collection.



**SYNOPSIS**

```
db.collectionspace.collection.remove([cond],[hint])
```

**CATEGORY**

Collection

**DESCRIPTION**

Delete records from the current collection.

cont (json object)

Selecting condition. If it is null, it will delete all the records. If it is not null, delete the records that match the condition.

hint (json object)

Specify access plan.

Format

The "cond" is a json object. When it is null (e.g. {} ), delete all records in the collection. "hint" is a json object that contains a single element. The field name of this element is ignored. But the value of the field in it is specified as the name of index which is needed to visit. When the value of field is null, it will visit all the records in the collection.

```
{["Field name 1":{"Match Operator 1":"value 1","Field name 2":{"Match Operator 2":"value 2"},...}],["":"index name" | null]}
```

**Examples**

Delete all the records in the collection "bar".

```
db.foo.bar.remove()
```

Delete records that match the condition in "cond" according to access plan.

```
db.foo.bar.remove ( {age: {$gte: 20}} , { " " : "myIndex" } )
```

This operation accesses all the records in the current collection according to the index named "myIndex", then it deletes all the records that match the condition(with a value greater than or equal to 20 in "age" field).

**db.collectionspace.collection.split()****NAME**

split - Split the records into different replica groups according to the given conditions in an environment with no less than two replica groups. This operation is a synchronous operation, so it will terminate and return after finishing the data split.

**SYNOPSIS**

```
db.collectionspace.collection.split(<source group>,<target group>,<percent(0~100)|  
cond>,[endcondition])
```

**CATEGORY**

Collection

**DESCRIPTION**

Split the records into different replica groups according to the given conditions in an environment with no less than two replica groups. This operation is a synchronous operation, so it will terminate and return after finishing the data split.

source group (string)

The source replica group.

target group (string)

The target replica group.

percent(0~100) (double)

percentage split condition. (Either "percent" or "condition".)

cond (json object)

Range split condition. (Either "condition" or "percent".)

endcondition (json object)

End of the range condition. (Optional, and it is effective when using range split, useless when using percentage split.)

Format

Data split is classified into range split and percentage split. Both "source group" and "target group" are common parameters in a type of string; both "condition" and "endcondition" are json objects that are needed when using range split; "percent" is a double float number that are needed when using percentage split.

Range split: When using range split, range partitions use precise condition, hash partitions use "number of partitions" as a condition. The starting point of split is a required condition, but the ending point is optional, in default, the ending point is the maximum data range of the split source.

("<source shard>", "<target shard>", <condition>)

Percentage split: db.foo.bar.split("<source shard>", "<target shard>", <percent>)



Note: When using range split, if the specified field of sharding key is in descending order, like: {groupingKey:{<Field 1>:<-1>}}, the starting point in "condition" or "partition" should be larger than the ending point.

**Examples**

Range-split of a hash-partition collection

```
db.foo.bar.split("shard1", "shard2", {Partition: 10}, {Partition: 20})
```

Range-split of range-partition collection

```
db.foo.bar.split("shard1", "shard2", {a: 10}, {a: 10000})
```

Percentage split

```
db.foo.bar.split("shard1", "shard2", 50)
```



Note: When using percentage split, please make sure that there are data in the source group, in other words, the collection is not empty.

**db.collectionspace.collection.splitAsync()****NAME**

splitAsync - This method has the same function as db.collection.collection.split, but this is a asynchronous operation, a task ID will be returned immediately after the task is created.

## SYNOPSIS

```
db.collection.space.collection. splitAsync(<source group>,<target group>,<percent(0~100)|
cond>,[endcondition])
```

## CATEGORY

Collection

## DESCRIPTION

This method has the same function as `db.collection.collection.split`, but this is an asynchronous operation, a task ID will be returned immediately after the task is created.

`db.collection.space.collection.update()`

## NAME

update - Update records in a collection.

## SYNOPSIS

```
db.collection.space.collection.update(<rule>,[cond],[hint])
```

## CATEGORY

Collection

## DESCRIPTION

Update records in a collection.

rule (json object)

Update rule. Records will be updated according the value of "rule".

cond (json object)

Selecting condition. If it is null, update all the records. If it is not null, it will update records that match the condition.

hint (json object)

Specify the access plan.

Format

The definition of `update()` must contain field "rule" which should be a json object. "cond" and "hint" are optional fields. Parameter "hint" is a json object that includes one field, the field name will be ignored, and its value specifies the index name, when the value of the field is null, all the records in the collection will be accessed, its format is `{"":null}` or `{"":<indexname>}`.

```
{< {<{""update operator 1"":{field name 1:"value"},"update operator 2":{field name
2": "value 2"},...}>,{field name 1":{match operator 1": "value 1"},field name 2":{match
operator 2": "value 2"},...}},{""index name" | null}}}
```



Note:

- `update()` does not support the update of sharding key in this version, if there are update operations on sharding key, these operations will be automatically neglected and the update of other fields will still work without any problems.

### Examples

Update all the records according to the update rule. That's to say, we merely set the value of "rule", but not "cond" or "hint".

```
db.foo.bar.update({$inc: {age: 1}})
```

This operation updates the field "age" in collection "bar" by adding 1 to the value of "age".

Select records that match the condition. Update these records according to update rules by setting the values of "rule" and "cond".

```
db.foo.bar.update({$unset: {age: ""}}, {age: {$exists: 1}, name: {$exists: 0}})
```

This operation will update records that contain the field "age" but not the field "name" in collection "bar" and delete the field "age" in these records.

Update records according to visiting plan, supposing that the collection contains the specified index name.

```
db.foo.bar.update({$inc: {age: 1}}, {age: {$gt: 20}}, {"": "testIndex"})>
```

This operation accesses records that contain a value which is no less than 20 in the field "age" according to the index named "testIndex"

## db.collectionspace.collection.upsert()

NAME

upsert - Update collection records.

### SYNOPSIS

```
db.collectionspace.collection.upsert(<rule>,[cond],[hint])
```

### CATEGORY

Collection

### DESCRIPTION

Update collection records. The methods upsert() and update() are both used to update records. But when no records is matched according to the parameter "cond", "update" does nothing, but "upsert" will insert data once.

rule (json object)

Update rule. Records will be updated according to the value of "rule".

cond (json object)

Selecting condition. When it is null, update all the records. If it is not null, update records that match the condition in "cond".

hint (json object)

Specify access plan.

Format

The definition of upsert() must contain field "rule" which should be a json object. "cond" and "hint" are optional fields. "hint" is a json object that include one field, the field name will be ignored, and its value specifies the index name, when the value of the field is null, all the records in the collection will be accessed, its format is {"":null} or {"": "<indexname>"}

```
{< {<""update operator 1"":{field name 1:"value"},update operator 2:{"field name 2":"value 2"},...}>,{{"field name 1":{"match operator 1":"value 1"},"field name 2":{"match operator 2":"value 2"},...}},[{"":"index name" | null}]}
```

**Note:**

- upsert() does not support the update of sharding key in this version, if there are update operations on sharding key, these operations will be automatically neglected and the update of other fields will still work without any problems.

**Examples**

Supposing that there are 2 records in the collection "bar".

```
{
  "_id": {
    "$oid": "516a76a1c9565daf06030000"
  },
  "age": 10,
  "name": "Tom"
}
{
  "_id": {
    "$oid": "516a76a1c9565daf06050000"
  },
  "a": 10,
  "age": 21
}
```

Update all the records according to the update rule. That's to say, we merely set the value of "rule", but not "cond" or "hint".

```
db.foo.bar.upsert({$inc: {age: 1}, $set: {name: "Mike"}})
```

This operation is equivalent to that of the method "update()". It updates all the records in the collection "bar". It adds 1 to the value of "age" and changes the value of "name" into "Mike". If a record doesn't contain the field "name", "\$set" will insert the field of "name" and its value into the record and return with the method "find".

```
{
  "_id": {
    "$oid": "516a76a1c9565daf06030000"
  },
  "age": 11,
  "name": "Mike"
}
{
  "_id": {
    "$oid": "516a76a1c9565daf06050000"
  },
  "a": 10,
  "age": 22,
  "name": "Mike"
}
```

Select records match the selecting condition, and update them according to the update rule. That's to say, we set the value of "rule" and "cond".

```
db.foo.bar.upsert({$inc: {age: 3}}, {type: {$exists: 1}})
```

This operation will update records that contains the field "type" in the collection "bar" and add 3 to the value of their "age" field. The 2 records above don't contain the field "type", so it will insert a new

record. This new record contains only the field "\_id" and the field "age". The value of "\_id" is automatically generated by the system. The value of "age" is 3.

```
{
  "_id": {
    "$oid": "516a76a1c9565daf06030000"
  },
  "age": 11,
  "name": "Mike"
}
{
  "_id": {
    "$oid": "516a76a1c9565daf06050000"
  },
  "a": 10,
  "age": 22,
  "name": "Mike"
}
{
  "_id": {
    "$oid": "516cfc334630a7f338c169b0"
  },
  "age": 3
}
```

Update records according to Access plan, supposing that the index name "testIndex" exists in the collection.

```
db.foo.bar.upsert({$inc: {age: 1}}, {age: {$gt: 20}}, {"": "testIndex"})>
```

This operation is equivalent to update(), it accesses records that contain a value which is no less than 20 in the field "age" according to the index named "testIndex", and add 1 to the value of the field "age". Then it returns:

```
{
  "_id": {
    "$oid": "516a76a1c9565daf06050000"
  },
  "a": 10,
  "age": 23,
  "name": "Mike"
}
```

## cursor.close()

NAME

close - Close the current cursor.

SYNOPSIS

```
cursor.close()
```

CATEGORY

Cursor

DESCRIPTION

Close the current cursor, the cursor is no longer available.

## Examples

Insert the 10 records.

```
for (i = 0; i < 10; i++) { db.foo.bar.insert ({a: i}) }
```

Query all records set "foo.bar".

```
var cur = db.foo.bar.find()
```

Use the cursor a record.

```
cur.next()
```

```
{
  "_id": {
    "$oid": "53b3c2d7bb65d2f74c000000"
  },
  "a": 0
}
```

Close cursor.

```
cur.close()
```

To obtain the next record.

```
cur.next()
```

No results are returned.

## cursor.current()

NAME

current - Return the record that current cursor points to.

SYNOPSIS

```
cursor.current()
```

CATEGORY

Cursor

DESCRIPTION

Return the record that current cursor points to.

## Examples

Select records with the value of "age" greater than 10 in the collection "bar" and return the record that current cursor points to.

```
db.foo.bar.find ({age: {$gt: 10}}).current()
```

## cursor.hint()

NAME

hint - Enumerate the result set according to the specified index.

SYNOPSIS

```
cousor.hint([hint])
```

## CATEGORY

Cursor

## DESCRIPTION

Enumerate the result set according to the specified index.

hint (json object)

It specifies the access plan to accelerate query speed.

Format

The method "cursor.hint()" contains parameter "hint". If it is not specified, the query will not use index to visit records. The parameter "hint" contains a one-field json object. The field name is ignored. But the field value is specified as the index which will be visited.

When it is null, it will visit all the records in the collection.

```
{"":null} or {"": "<indexname>"}
```

## Examples

Enumerate the records contain field "age" in the collection "bar" according to index "ageIndex" and return the result set.

```
db.foo.bar.find({age: {$exists: 1}}).hint({"": "ageIndex"})
```

## cursor.limit()

### NAME

limit - Limit the maximum number of returned records.

### SYNOPSIS

```
cursor.limit([num])
```

## CATEGORY

Cursor

## DESCRIPTION

Limit the maximum number of returned records.

num (int)

The parameter "num" specifies the maximum number of returned records.

Format

The method "cousor.limit()" contains the parameter "num". It is in the type of "int". If the value of "num" is not specified, it will return all the records in the result set. If only the first 5 records are wanted, the value of "num" should be set as 5.

## Examples

Select records with the value of "age" greater than 10 in the collection "bar" and return the first 10 records.

```
db.foo.bar.find({age: {$gt: 10}}).limit(10)
```



**Note:**

If the amount of records in the result set is lesser than 10, it will return all the records. If it is greater than 10, it will return the first 10 records.

**cursor.next()****NAME**

next - Return the next record of the record that current cursor points to.

**SYNOPSIS**

```
cursor.next()
```

**CATEGORY**

Cursor

**DESCRIPTION**

Return the next record of the record that current cursor points to.

**Examples**

Select records with the value of "age" greater than 10 in the collection "bar" and return the next record of the record that current cursor points to.

```
db.foo.bar.find({age: {$gt: 10}}).next()
```

**cursor.size()****NAME**

size - Return the amount of records from current cursor to final cursor.

**SYNOPSIS**

```
cursor.size()
```

**CATEGORY**

Cursor

**DESCRIPTION**

Return the amount of records from current cursor to final cursor.

**Examples**

Select records with the value of "age" greater than 10 in the collection "bar" and return the amount of records in the range from current cursor to final cursor.

```
db.foo.bar.find({age: {$gt: 10}}).size()
```

## cursor.skip()

### NAME

skip - It is used to specify the first returned record In the result set.

### SYNOPSIS

```
cursor.skip([num])
```

### CATEGORY

Cursor

### DESCRIPTION

It is used to specify the first returned record In the result set.

num (int)

Specify the index number of first returned record in the result set.

Format

The method "cousor.skip()" contains the parameter "num". It is in the type of "int". If the value of "num" is null or 0, the method will return the whole result set. If you want to get a result set that begins with the 3rd record, you should set the value of "num" as "2".

### Examples

Select records with the value of "age" greater than 10 and return records which begin with the 5th record. That's to say, it will skip the first 4 records.

```
db.foo.bar.find({age: {$gt: 10}}).skip(4)
```



### Note:

If the amount of records in a result set is lesser than 5, no record will be returned. If the amount of records in a result set is greater than 5, it will return all the records after the fourth one..

## cursor.sort()

### NAME

sort - Sort the result set according to a specified field.

### SYNOPSIS

```
cousor.sort([sort])
```

### CATEGORY

Cursor

### DESCRIPTION

Sort the result set according to a specified field.

sort (json object)

Sort the result set according to a specified field. The value of it is 1 or -1. 1 represents ascending order. -1 represents descending order.

### Format

The method "cursor.sort()" contains a parameter called "sort". It is a json object. If it is not specified, result set will not be sorted.

{<Field name 1>:<-1 | 1>,<Field name 2>:<-1 | 1>,...}

### Examples

Return records that contains the value of "age" greater than 20. And it just return the fields named "name" and "age" sorted on the field "age" in ascending order.

```
db.foo.bar.find({age: {$gt: 20}}, {age: "", name: ""}).sort({age: 1})
```



#### Note:

Through the method "find()", we can select the fields we need. In the example above, it returns "age" and "name", so if we use "sort()" to sort the records, we can only sort the result set on "age" and "name". If we don't specify the parameter "sel" in the method "find", we can sort the result set on any field.

## cursor.toArray()

### NAME

toArray - Return result set in the type of array.

### SYNOPSIS

```
cursor.toArray()
```

### CATEGORY

Cursor

### DESCRIPTION

Return result set in the type of array.

### Examples

Select records with the value of "age" greater than 5 in the collection "bar" return the result set in the type of array.

```
db.foo.bar.find({age: {$gt: 10}}).size()
```

```
Return: {
  "_id": {
    "$oid": "516a76a1c9565daf06030000"
  },
  "age": 10,
  "name": "Tom"
}, {
  "_id": {
    "$oid": "516a76a1c9565daf06050000"
  },
  "age": 20,
  "a": 10
}, {
  "_id": {
    "$oid": "516a76a1c9565daf06040000"
  },
  "age": 15
```

```
}
```

## rg.createNode()

NAME

createNode - Create a node in a replica group.

SYNOPSIS

```
rg.createNode(<host>,<service>,<dbpath>,[config])
```

CATEGORY

ReplicaGroup

DESCRIPTION

Create a node in a replica group.

host (string)

Specify the host name of the node.

service (string)

Node port number.

dbpath (string)

Node path.

config (json)

Node's configuration information, such as, the size of configuration log, whether enable transaction function or not. For more details, please check DATABASE CONFIGURATION. Format

The method "rg.createNode()" contains 4 parameters: Host, Service, dbPath, config. The first three parameters in the above list are required and in the type of string, the last one is an optional json object configuration option.

```
("<Host>","<service>","<dbpath>",[{"<configParam>:value,...}])
```

Examples

Create a node with the port of 11800 in a replica group named "group", and specify the size of the log file as 64 MB.

```
rg.createNode("vmsvr2-suse-x64", 11800, "/opt/sequoiadb/data/11800", {logfilesz: 64})
```



Note:

More than one nodes can be created within one replica group.

## rg.getDetail()

NAME

getDetail - Return the information of the current replica group.

SYNOPSIS

```
rg.getDetail()
```

## CATEGORY

## ReplicaGroup

## DESCRIPTION

Return the information of the current replica group.

## Examples

```
> rg.getDetail()
{
  "Group": [
    {
      "dbpath": "/opt/sequoiadb/data/11800",
      "HostName": "vmsvr2-suse-x64",
      "Service": [
        {
          "Type": 0,
          "Name": "11800"
        },
        {
          "Type": 1,
          "Name": "11801"
        },
        {
          "Type": 2,
          "Name": "11802"
        },
        {
          "Type": 3,
          "Name": "11803"
        }
      ],
      "NodeID": 1000
    },
    {
      "dbpath": "/opt/sequoiadb/data/11850",
      "HostName": "vmsvr2-suse-x64",
      "Service": [
        {
          "Type": 0,
          "Name": "11850"
        },
        {
          "Type": 1,
          "Name": "11851"
        },
        {
          "Type": 2,
          "Name": "11852"
        },
        {
          "Type": 3,
          "Name": "11853"
        }
      ],
      "NodeID": 1001
    }
  ],
  "GroupID": 1001,
  "GroupName": "group",
  "PrimaryNode": 1001,
```

```

    "Role": 0,
    "Status": 1,
    "Version": 3,
    "_id": {
      "$oid": "517b2fc33d7e6f820fc0eb57"
    }
  }
}

```

## rg.getMaster()

NAME

getMaster - Return the master node of a replica group.

SYNOPSIS

```
rg.getMaster()
```

CATEGORY

ReplicaGroup

DESCRIPTION

Return the master node of a replica group.

Examples

Return the master node of a replica group.

```

> rg.getMaster()
Return: vmsvr2-suse-x64:11850

```

## rg.getNode()

NAME

getNode - Return the information of a specified node.

SYNOPSIS

```
rg.getNode(<nodename|hostname>,<servicename>)
```

CATEGORY

ReplicaGroup

DESCRIPTION

Return the information of a specified node.

nodename (string)

Node name. Either "nodename" or "hostname".

hostname (string)

Hostname. Either "hostname" or "nodename".

servicename (string)

Server name.

Format

`rg.getNode()` has two parameters, the first parameter can be either the node name or host name, the second parameter is the server name. These parameters are all required and must in the type of string.

("<Node name>|<hostname>",<server name>")

#### Examples

Return a node with specified hostname and server name.

```
> rg.getNode("vmssvr2-suse-x64", "11800")
Return: vmssvr2-suse-x64:11800
```

## rg.getSlave()

### NAME

`getSlave` - Return a random slave node in the current replica group.

### SYNOPSIS

```
rg.getSlave()
```

### CATEGORY

ReplicaGroup

### DESCRIPTION

Return a random slave node in the current replica group.

#### Examples

Return a random slave node in the current replica group:

```
> rg.getSlave()
Return: vmssvr2-suse-x64:11800
```

## rg.removeNode()

### NAME

`removeNode` - Delete the specified node in the replica group

### SYNOPSIS

```
rg.removeNode(<host>,<service>,[config])
```

### CATEGORY

ReplicaGroup

### DESCRIPTION

Delete the specified node in the replica group

`host` (string)

Node's hostname.

`service` (string)

Node's port number.

**config (json)**

Node's configuration information.

Format

Definition Format `rg.removeNode()` method has three parameters: host, service, config, as shown above, in the following format:

("<hostname>","<port number>[{<configParam>:value,...}])

**Examples**

Delete specified node from the replica group.

```
rg.removeNode("vmsvr2-suse-x64", 11800)
```

**Note:**

The specified node to be deleted must exist, otherwise an exception will occur.

**rg.start()**

NAME

start - Start the replica group.

SYNOPSIS

```
rg.start()
```

CATEGORY

ReplicaGroup

DESCRIPTION

Start the replica group. After a replica group is started, operations like creating nodes can be executed. Users can start specified node with the method `db.startRG(<name>)`.

**Examples**

Start replica group:

```
rg.start() //Equivalent to db.startRG("group")
```

**rg.stop()**

NAME

stop - Stop the current replica group.

SYNOPSIS

```
rg.stop()
```

CATEGORY

ReplicaGroup



**DESCRIPTION**

Stop the current replica group. Operations on the replica group like creating a node in this group cannot be done after the replica group is stopped.

**Examples**

Stop the current replica group.

```
rg.stop()
```

**node.connect()****NAME**

connect - Connect the database to the current node.

**SYNOPSIS**

```
node.connect()
```

**CATEGORY**

Node

**DESCRIPTION**

Connect the database to the current node. After connecting them, users can execute a set of operations. Users can check relative manipulations with the method "node.connect().help()".

**Examples**

Connect the database to the current node.

```
> node.connect()
Return: vmsvr2-suse-x64:11800
```

**node.getHostName()****NAME**

getHostName - Return the hostname of a node.

**SYNOPSIS**

```
node.getHostName()
```

**CATEGORY**

Node

**DESCRIPTION**

Return the hostname of a node.

**Examples**

Return the hostname of the current node.

```
> node.getHostName()
```

Return: vmsvr2-suse-x64

## node.getNodeDetail()

### NAME

getNodeDetail - Return the information of the current node.

### SYNOPSIS

```
node.getNodeDetail()
```

### CATEGORY

Node

### DESCRIPTION

Return the information of the current node.

### Examples

Return the information of the current node.

```
> node.getNodeDetail()
Return: 1000: vmsvr2-suse-x64: 11800 (group)
Here "1000" is node ID(NodeID); "vmsvr2-suse-x64" is hostname(HostName); "11800" is service
name(ServiceName), "(group)" is the ReplicaGroup name of the node.
```

## node.getServiceName()

### NAME

getServiceName - Return the server name of a node.

### SYNOPSIS

```
node.getServiceName()
```

### CATEGORY

Node

### DESCRIPTION

Return the server name of a node.

### Examples

Return of server name of the current node.

```
> node.getServiceName()
Return: 11800
```

## node.start()

### NAME

start - Start the current node.

**SYNOPSIS**

```
node.start()
```

**CATEGORY**

Node

**DESCRIPTION**

Start the current node.

**Examples**

Start a current node named "node".

```
node.start()
```

**node.stop()****NAME**

stop - Stop the current node.

**SYNOPSIS**

```
node.stop()
```

**CATEGORY**

Node

**DESCRIPTION**

Stop the current node.

**Examples**

Stop the current node.

```
node.stop()
```

**domain.alter()****NAME**

alter - Alter the specified fields of the current domain.

**SYNOPSIS**

```
domain.alter(<options>)
```

**CATEGORY**

Domain

**DESCRIPTION**

Alter the specified fields of the current domain.

**options (Json Object)**

Need to modify the list of attributes.

Groups

The contained replica groups.

Groups:['data1','data2']

AutoSplit

Automatically splitted.

AutoSplit:true | false



Note:

- Make sure that there is no data in the domain before deleting it.
- The modification of the "AutoSplit" field will not affect the former created collections and collection spaces.

**Examples**

Create a domain contains two replica groups, and then enable it to auto-split.

```
var domain = db.createDomain('mydomain', ['data1', 'data2'], {AutoSplit: true})
```

Delete a replica group from a domain, and then add another one.

```
domain.alter({Groups: ['data1', 'data3']})
```

**domain.listCollections()**

NAME

listCollections - Enumerate collections.

SYNOPSIS

```
domain.listCollections()
```

CATEGORY

Domain

DESCRIPTION

Enumerate collections. It will return the information of every collection in a domain.

**Examples**

```
domain.listCollections()
```

Return:

```
{
  "Name": "foo.bar"
}
```

**domain.listCollectionSpaces()**

NAME

listCollectionSpaces - The collection spaces in the domain.

**SYNOPSIS**

```
domain.listCollectionSpaces()
```

**CATEGORY**

Domain

**DESCRIPTION**

The collection spaces in the domain.

**Examples**

```
domain.listCollectionSpace()
```

Returns:

```
{
  "Name": "foo"
}
```

## Operator

---

**Matching records**

Match	Description	Sample
<a href="#">\$gt</a>	Greater than	db.foo.bar.find({age:{>:20}})
<a href="#">\$gte</a>	Greater than or equal to	db.foo.bar.find({age:{>=:20}})
<a href="#">\$lt</a>	Lesser than	foo.bar.find({age:{<:20}})
<a href="#">\$lte</a>	Lesser than or equal to	db.foo.bar.find({age:{<=:20}})
<a href="#">\$ne</a>	Unequal to	db.foo.bar.find({age:{<ne>:20}})
<a href="#">\$et</a>	equal to	db.foo.bar.find({age:{<et>:20}})
<a href="#">\$mod</a>	modulo	db.foo.bar.find({age:{<mod>:[6,5]}})
<a href="#">\$in</a>	Exist in the collection	db.foo.bar.find({age:{<in>:[20,21]}})
<a href="#">\$isnull</a>	is "null" or does not exist.	Db.foo.bar.find({<age>:{<isnull>:1}})
<a href="#">\$nin</a>	Not exist in the collection	db.foo.bar.find({age:{<nin>:[20,21]}})
<a href="#">\$all</a>	All	db.foo.bar.find({age:{<all>:[20,21]}})
<a href="#">\$and</a>	And	db.foo.bar.find({<\$and>:{age:20},{name:"Tom"}})
<a href="#">\$not</a>	Not	db.foo.bar.find({<\$not>:{age:20},{name:"Tom"}})
<a href="#">\$or</a>	Or	db.foo.bar.find({<\$or>:{age:20},{name:"Tom"}})
<a href="#">\$type</a>	Data type	db.foo.bar.find({age:{<\$type>:16}})
<a href="#">\$exists</a>	Exists	db.foo.bar.find({age:{<\$exists>:1}})
<a href="#">\$elemMatch</a>	Match elements	db.foo.bar.find({age:{<\$elemMatch>:20}})
<a href="#">\$+Identifier</a>	Matching array elements	db.foo.bar.find({"array.\$2":10})
<a href="#">\$size</a>	Size	db.foo.bar.find({array:{<size>:3}})
<a href="#">\$regex</a>	Regular	db.foo.bar.find({str:{<\$regex>:'dh.*fj',\$options:'i'}})

## Update rule

Update operator	Description	Information	Sample
<code>\$inc</code>	Add	Add specified value to the value of specified field.	<code>db.foo.bar.update({\$inc:{age:25}})</code>
<code>\$set</code>	Set	Set the value of specified field as specified value.	<code>db.foo.bar.update({\$set:{age:10}})</code>
<code>\$unset</code>	Delete	Delete specified field in an object.	<code>db.foo.bar.update({\$unset:{age:''}})</code>
<code>\$addto</code>	Add to set	If the element to add doesn't exist in the array, the command add it, or it will be ignored. The target field should be array.	<code>db.foo.bar.update({\$addto: {array:[3,4,5]}})</code>
<code>\$pop</code>	Delete the last N value in an array	Delete the last N value in an array. The target field should be array.(If N is lesser than 0, it will delete the first N elements in the array.)	<code>db.foo.bar.update({\$pop:{array:2}})</code>
<code>\$pull</code>	Delete value	Delete the specified value in a target array. The target field should be array.	<code>db.foo.bar.update({\$pull:{array:2}})</code>
<code>\$pull_all</code>	Delete array	Delete all the values of a specified array in a target array. The target field should be array.	<code>db.foo.bar.update({\$pull_all:{array: [2,3,4]}})</code>
<code>\$push</code>	Push value	Insert a value into a target array. The target field should be array.	<code>db.foo.bar.update({\$push: {array:2}})</code>
<code>\$push_all</code>	Push array	Insert all the values in a specified array into a target array. The target field should be array.	<code>db.foo.bar.update({\$push_all: {array:[2,3,4]}})</code>

## Aggregation operator

Parameter name	Description	Example
<code>\$project</code>	Select the field name to be output, "1" indicates that the output "0" means no output, you can also implement fields.	<code>{ \$project: { field1:1, field:0, alias: "\$field3" } }</code>
<code>\$match</code>	Select the matching criteria to achieve from a collection of records, quite where the SQL statement.	<code>{ \$match: { field: { \$lte: value } } }</code>
<code>\$limit</code>	Limit the number of records returned.	<code>{ \$limit: 10 }</code>
<code>\$skip</code>	Start point control of the result set, the result set that skips the number of records specified.	<code>{ \$skip: 5 }</code>
<code>\$group</code>	Achieve grouping of records, similar to SQL's group by statement, "_id" designated group field.	<code>{ \$group: { _id: "\$field" } }</code>
<code>\$sort</code>	Achieve the sort of result set "1" represents Ascending, "-1" for descending.	<code>{ \$sort: { field1:1, field2:-1,... } }</code>

`$group` Aggregation operator supports the following aggregate functions:

Function name	Description
<code>\$addToSet</code>	Adds the specified field values in the array, the same field value will be added once.
<code>\$first</code>	Take packet field values in the first record.

Function name	Description
<code>\$last</code>	Take packet field value in the record.
<code>\$max</code>	Take the maximum value of the packet field.
<code>\$min</code>	Take the smallest packet field values.
<code>\$avg</code>	Take the average packet field values.
<code>\$push</code>	Add all the fields to the array, even if the same array of field values that already exist, and continue to add.
<code>\$sum</code>	Take the sum of packet field values.



Note:

In SequoiaDB, the structure called JSON (field name: value) is used to express "=". Of course, you can use `$set` operator.

Sample:

```
db.collection.space.collection.find({a:42})
```

This command find records that contains the value of "a" equal to 42.

## Match Operator

operator	description	samples
<code>\$gt</code>	greater than	<code>db.foo.bar.find({age:{&gt;:20}})</code>
<code>\$gte</code>	great than or equal	<code>db.foo.bar.find({age:{&gt;=:20}})</code>
<code>\$lt</code>	less than	<code>foo.bar.find({age:{&lt;:20}})</code>
<code>\$lte</code>	less than or equal	<code>db.foo.bar.find({age:{&lt;=:20}})</code>
<code>\$ne</code>	not equal	<code>db.foo.bar.find({age:{&lt;ne:20}})</code>
<code>\$in</code>	exist in collection	<code>db.foo.bar.find({age:{in:[20,21]}})</code>
<code>\$nin</code>	not exist in collection	<code>db.foo.bar.find({age:{nin:[20,21]}})</code>
<code>\$all</code>	all	<code>db.foo.bar.find({age:{all:[20,21]}})</code>
<code>\$and</code>	and	<code>db.foo.bar.find({&amp;:[age:20,{name:"Tom"}]})</code>
<code>\$not</code>	not	<code>db.foo.bar.find({\$not:{age:20},{name:"Tom"}})</code>
<code>\$or</code>	or	<code>db.foo.bar.find({\$or:[age:20,{name:"Tom"}]})</code>
<code>\$type</code>	date type	<code>db.foo.bar.find({age:{type:16}})</code>
<code>\$exists</code>	exist	<code>db.foo.bar.find({age:{exists:1}})</code>
<code>\$elemMatch</code>	element match	<code>db.foo.bar.find({age:{elemMatch:20}})</code>
<code>\$+Identifier</code>	Matching array elements	<code>db.foo.bar.find({"array.\$2":10})</code>
<code>\$size</code>	size	<code>db.foo.bar.find({array:{size:3}})</code>
<code>\$regex</code>	regex	<code>db.foo.bar.find({str:{regex:'dh.*fj',\$options:'i'}})</code>

`$gt`

Grammar

```
{<field name>: {$gt:<value>}}
```

Description

"\$gt" finds records that contain value greater than the specific value "value" on the field "field name".

## Sample

- Find records that contains value greater than 20 on the field "age" in the collection "bar" of the collection space "foo".

```
db.foo.bar.find({age: {$gt: 20}})
```

- When "\$gt" manipulates records with nested objects, the function `update()` will update records that contains "age" value of 25 with records which contains a "type" value greater than 2 under the field "service".

```
db.foo.bar.update({$set: {age: 25}}, {"service.type": {$gt: 2}})
```

## \$gte

## Grammar

```
{<field name>: {$gte: <value>}}
```

## Description

"\$gte" finds records that contain value greater than or equivalent to the specific value "value" on the field "field name".

## Sample

- Find records that contains value greater than or equivalent to 20 on the field "age" in the collection "bar" of the collection space "foo".

```
db.foo.bar.find({age: {$gte: 20}})
```

- When "\$lte" manipulates records with nested objects, the function `update()` will update records that contains "age" value of 25 with records which contains a "type" value greater than or equivalent to 2 under the field "service".

```
db.foo.bar.update({$set: {age: 25}}, {"service.type": {$gte: 2}})
```

## \$lt

## Grammar

```
{<field name>: {$lt: <value>}}
```

## Description

"\$lte" finds records that contain value lesser than the specific value "value" on the field "field name".

## Sample

- Find records that contains value lesser than 20 on the field "age" in the collection "bar" of the collection space "foo".

```
db.foo.bar.find({age: {$lt: 20}})
```

- When "\$lte" manipulates records with nested objects, the function `update()` will update records that contains "age" value of 25 with records which contains a "type" value lesser than 15 under the field "service".

```
db.foo.bar.update({$set: {age: 25}}, {"service.type": {$lt: 15}})
```

## \$lte

## Grammar

```
{<field name>: {$lte: <value>}}
```



## Description

"\$lte" finds records that contain value lesser than or equivalent to the specific value "value" on the field "field name".

## Sample

- Find records that contains value lesser than or euqivalent to 20 on the field "age" in the collection "bar" of the collection space "foo".
- When "\$lte" manipulates records with nested objects, the function `update()` will update records that contains "age" value of 25 with records which conatins a "type" value lesser than or euqivalent to 15 under the field "service".

```
db.foo.bar.find({age: {$lte: 20}})
```

```
db.foo.bar.update({$set: {age: 25}}, {"service.type": {$lte: 2}})
```

## \$ne

## Grammar

```
{<field name>: {$ne: <value>}}
```

## Description

"\$ne" finds records that contain value unequal to the specific value "value" on the field "field name".

## Sample

- Find records that contains value unequal to 20 on the field "age" in the collection "bar" of the collection space "foo".
- When "\$lte" manipulates records with nested objects, the function `update()` will update records that contains "age" value of 25 with records which conatins a "type" value unequal to 2 under the field "service".

```
db.foo.bar.find({age: {$ne: 20}})
```

```
db.foo.bar.update({$set: {age: 25}}, {"service.type": {$ne: 15}})
```

## \$et

## Grammar

```
{<field name>: {$et: <value>}}
```

## Description

"\$et" finds records that contain value equal to the specific value "value" on the field "field name".

## Sample

- Find records that contains value equal to 20 on the field "age" in the collection "bar" of the collection space "foo".
- When "\$et" manipulates records with nested objects, the function `update()` will update records that contains "age" value of 25 with records which conatins a "type" value equal to 15 under the field "service".

```
db.foo.bar.find({age: {$et: 20}})
```

```
db.foo.bar.update({$set: {age: 25}}, {"service.type": {$et: 15}})
```

## \$mod

### Grammar

```
{<field>: {$mod: [value1,value2]},...}
```

### Description

\$mod is a modulo operator, to return the records which the value of specified field count modulo value1 is equal to value2.



#### Note:

1. Param value1 is a integer except 0; if it is filled with float, then only the integer part is effective; in addition, it can't be other types.
2. Param value2 is a integer; if it is filled with float, then only the integer part is effective; other types are treated with 0.

### Sample

- Return the records that the value of field age count modulo 5 is equal to 3 in collection bar

```
db.foo.bar.find({age: {$mod: [5, 3]}})
return
{
  "_id": {
    "$oid": "521d5446e2d3c4e31c000000"
  },
  "age": 3
}...
```

```
db.foo.bar.find({age: {$mod: [2.3, 1.5]}})
return
{
  "_id": {
    "$oid": "521d5446e2d3c4e31c000000"
  },
  "age": 3
}
{
  "_id": {
    "$oid": "521d544ee2d3c4e31c000002"
  },
  "age": 5
}...
```

Only the integer part are valid for the two elements in the array[2.3,1.5] .

## \$in

### Grammar

```
{<field name>: {$in: [<value 1>,<value 2>,...<value N>]}}
```

### Description

"\$in" finds records that match any value of the array ([<value 1>,<value 2>,...<value N>]) on the field ("field name") within a collection. If the field ("field name") is in the type of array, records that match any value of the array ([<value 1>,<value 2>,...<value N>]) on the field ("field name").

## Sample

- Find records with "age" value of 20 or 25 in the collection "bar":

```
db.foo.bar.find({age: {$in: [20, 25]}})
```

- When manipulating "\$in" on array fields, the following expression finds records with "name" value of "Tom" or "Mike" in the collection "bar", then deletes the field of "age" in these records.

```
db.foo.bar.update({$unset: {age: ""}}, {name: {$in: ["Tom", "Mike"]}})
```



Note: Array containing one value like {<field name>:{\$in:[<value>]}} equals {<field name>:<value>}.

```
db.foo.bar.find({age: {$in: [20]}}) is equivalent to db.foo.bar.find({age: 20})
```

## \$isnull

## Syntax

```
{<field name>: {$isnull: <0|1>}}
```

## Description

Select collection at the specified "<field name>" is empty or does not exist. "0" represents the field exists and is not expected "null", "1" represents the hope that the field does not exist or is "null".

## 示例

- Select Collection "bar" medium "age" field is not empty and there is a record.

```
db.foo.bar.find({age: {$isnull: 0}})
```

- Select Collection "bar" nested object "content.name" does not exist or is null record.

```
db.foo.bar.find({"content.name": {$exists: 1}})
```

## \$nin

## Grammar

```
{<field name>: {$nin: [<value 1>, <value 2>, ...<value N>]}}
```

## Description

"\$nin" finds records that match none of the values in the array ([<value 1>, <value 2>, ...<value N>]) on the field ("field name") within a collection. If the field ("field name") is in the type of array, records containing elements that match any value of the array ([<value 1>, <value 2>, ...<value N>]) on the field ("field name").

## Sample

- Find records without the field "age" or the value of 20 or 25 on the field "age" in the collection "bar":

```
db.foo.bar.find({age: {$nin: [20, 25]}})
```

- When manipulating "\$nin" on array fields, the following expression finds records without the field "name" or the value of "Tom" or "Mike" on the field "name" in the collection "bar", then deletes the field of "age" in these records.

```
db.foo.bar.update({$unset: {age: ""}}, {name: {$nin: ["Tom", "Mike"]}})
```



Note:

Array containing one value like {<field name>:{\$in:[<value>]}} equals {<field name>:<value>}.

```
db.foo.bar.find({age: {$nin: [20]}}) equals to db.foo.bar.find({age: {$ne: 20}})
```

## \$all

### Grammar

```
{<Field name>: {$all: [<value 1>, <value 2>, ... <value N>]}}
```

### Description

Manipulation "\$all" is manipulated on field names in type of Array. It will find records that contains all the values in the specified array ([<value 1>, <value 2>, ... <value N>]) on the specified field ("<field name>").

### Sample

- Find records that contain "Tom" and "Mike" on the specified field "name" in the collection "bar":

```
db.foo.bar.find({name: {$all: ["Tom", "Mike"]}})
```

As a result, the system will match records that contains the field "name" and "name" values as follow in the collection "bar":

```
["Tom", "Mike", ..]  
["Tom", "Jhon", "Mike", ...]
```

But the record below won't be matched:

```
["Tom", "Jhon"]
```



#### Note:

When "\$all" is manipulated on a field not in type of array, for example: "db.foo.bar.find({age:{\$all:[20]}})" equals to "db.foo.bar.find({age:20})".

## \$and

### Grammar

```
{$and: [{<expression 1>}, {<expression 2>}, ..., {<expression N>}]}
```

### Description

\$and is a logical manipulation. It is aimed at searching for records that satisfies all the expressions (<expression 1> , <expression 2>, ..., <expression N>).

But if the result of the 1st expression (<expression 1>) is false, SequoiaDB will ignore all the expressions afterward.

### Sample

- Find records with "age" value of 20 and "price" value of less than 10 in collection "bar":

```
db.foo.bar.find({$and: [{age: 20}, {price: {$lt: 10}}]})
```



#### Note:

Sequoiadb provides implicit "and" manipulation: separeate expressions with ",". For example,

```
db.foo.bar.find({age: 20, price: {$lt: 10}})
```

- When "and" is manipulated on the same field like {age : {\$lt:20}}and{age:{\$exists:1}}, then we can use "\$and" to manipulate the two separated expressions, or combine them: {age:{\$lt:20,\$exists:1}}.

```
db.foo.bar.update({$inc: {salary: 200}}, {$and: [{age: {$lt: 20}}, {age: {$exists: 1}}]})  
db.foo.bar.update({$inc: {salary: 200}}, {age: {$lt: 20, $exists: 1}})
```

When the results of 2 manipulations are the same, it will find records containing the field "age" with the "age" value of less than 20 in collection bar, then add 200 to the "salary" value of these records.

## \$not

### Grammar

```
{ $not: [ {<expression 1>}, {<expression 2>}, ..., {<expression N>} ] }
```

### Description

The operation "\$not" is a logical "not". Its role is choose not to match expression(<expression 1><expression 2>,...,<expression N>) records. As long as it does not meet any one of these expressions, recording will return.

### Sample

- Select records that contains the value of "age" unequal to 20 or the value of "price" equal to or greater than 10.

```
db.foo.bar.find( { $not: [ {age: 20}, {price: { $lt: 10 }} ] } )
```

## \$or

### Grammar

```
{ $or: [ {<expression 1>}, {<expression 2>}, ..., {<expression N>} ] }
```

### Description

The operation "\$or" is a logical "or" operation. It returns "true" when a record matches one expression of the expression group (<expression 1>,<expression 2>,...,<expression N>).

Once the result is "true", the matched record will be returned.

### Sample

- Select records that contain the value "Tom" on the field "name", and the value 20 on the field "age" or the value of "price" lesser than 10.

```
db.foo.bar.find( { name: "Tom", $or: [ {age: 20}, {price: { $lt: 10 }} ] } )
```

- Apply "\$or" in nested fields. The following command will select records that contains the value of "age" lesser than 20 or the value "system" on the field "type" which is nested in the field "snapshot", and use [\\$inc](#) to update the value of "salary" in these records.

```
db.foo.bar.update( { $inc: {salary: 200} }, { $or: [ {age: { $lt: 20 }} ], { "snapshot.type": "system" } } )
```

## \$type

### Grammar

```
{<field name>: { $type $type: <BSON type> } }
```

### Description

Match the records which the value of the "<field name>" equals to the specified value of "<BSON type>" in the collection.

## BSON Type

Type	Description	value
32-bit integer	Integer,range -2147483648 to 2147483647	16
64-bit integer	Long Integer,range -9223372036854775808 to 9223372036854775807.  if the users specify a value can't be applied to an Integer,then SequoiaDB will converted to a Long Integer automatically.	18
double	Float,range 1.7E-308 to 1.7E+308	1
string	String	2
ObjectID	12Byte object ID	7
boolean	Boolean(true false)	8
date	Date(YYYY-MM-DD )	9
timestamp	Timestamp(YYYY-MM-DD-HH.mm.ss.ffffff )	17
Binary data	Base64 binary form	5
Regular expression	Regular Expression	11
Object	Nested JSON Document Object	3
Array	Nested Array Object	4
null	Null	10

## Samples

- Select records which the type of age field is Integer .  
`db.foo.bar.find({age: {$type: 16}})`
- Select records which the type of nested field arr is Array.  
`db.foo.bar.find({"content.arr": {$type: 4}})`

## \$exists

## Grammar

```
{<field name>:{$exists:<0|1>}}
```

## Description

"\$exists" finds records that contain or do not contain the field "field name". "0" means finding records that do not contain the field "field name". "1" means finding records that contain the field "field name".

## Sample

- Find records that contain the field "age" in collection "bar".  
`db.foo.bar.find({age: {$exists: 1}})`
- Find records that do not contain object "content" with the field "name" in collection "bar".  
`db.foo.bar.find({"content.name": {$exists: 0}})`

## \$elemMatch

## Grammar

```
{<field>:{$elemMatch:<value>}}
```

## Description

return all record in collection where the '<field name>' match the specified '<value>'.

## Sample

- array object match

```
db.foo.bar.find({"add.0": {$elemMatch: "china"}, "add.1": {$elemMatch: "ABC"}})
```

return all records in the collection bar where the value of first element is "china" and the value of second element is "ABC" in the array add.

- json object match

```
db.foo.bar.find({content: {$elemMatch: {name: "Tom", phone: 123}}})
```

field content is a nested json object, this operation is to match all records in the collection bar where the field name has the value "Tom" and the field phone has the value "123" in the nested object content.

## \$+Identifier

## Grammar

```
{"field name.$+Identifier": value}
```

## Description

\$+Identifier is a special command character, this command breaks act only on array of object , identifier is an integer, such as \$1,\$3, the equivalent of a temporary storage identifier, will match the success of indexed array elements stored. The following is wrong to write format: \$5.4, \$a2, \$3c, \$MA.

This command only affects the character array, the array is used instead of the index Key and can be matched to pass an index value of the first parameter to the method update [update](#) of the rule.

## Example

- Inquiry

Record: {a:[1,2,3,4,5]};{a:[1,4,5]};{a:[4,2,1]} now want to check out the array element exists record 5, use the following command.

```
db.foo.bar.find({"a.$1": 5}, {a: 1})
```

As long as the existence of a record array object element 5, will be able to return. Return the following results:

```
{ "a": [ 1, 4, 5 ] }
{ "a": [ 1, 2, 3, 4, 5 ] }
```

- Update

1. Record { a : [ 1, 2, 3, 4, 5 ] }, and now you want to modify a array of elements, the elements of the value 4 into 100, use the following command.

```
db.foo.bar.update ({$set: {"a.$1": 100}}, {"a.$1": 4})
```

When matching element index Key 4 is 3, so the update rules { "\$set" : { "a.\$1": 100 } }, and \$1 is 3, the system automatically converts the update rules to { "\$set" : { "a.3" : 100 } }

After updating the record as follows:

```
{ a : [ 1, 2, 3, 100, 5 ] }
```

2. Record { a : [ 1, 2, 3, 4, 5 ], b : [ 6, 7, 8 ] }, now want to modify the elements of the array a, the element value was changed 4 to 100, and the elements of the array b changes in value of 6 to 200, use the following command.

```
db.foo.bar.update({ "$set" : { "a.$1" : 100, "b.$2" : 200 } }, { "a.$1": 4, "b.$2" : 6 })
```

After updating the record as follows:

```
{ a : [ 1, 2, 3, 100, 5 ], b : [ 200, 7, 8 ] }
```



Note: If there are multiple elements comply with the rules, then only modify the first. In the following example:

3. Record { a : [ 1, 2, 2, 2, 5 ] }, now want to modify a array of elements, the elements changed to a value of 2 to 100, use the following command.

```
db.foo.bar.update({ "$set" : { "a.$1" : 100 } }, { "a.$1": 2 })
```

After updating the record as follows:

```
{ a : [ 1, 100, 2, 2, 5 ] }
```

## \$size

### Grammar

```
{"<field name>":{"$size":"<value>"}}
```

### Description

By applying the operator "\$size", users can find records that contain an array field named "field name", the length of which should be equal to "value".

### Sample

- Return records that contain the array field "arr"

```
db.foo.bar.find({arr:{$size:2}})
```

## \$regex

### description

\$regex Operation provides regular expression pattern matching string search functions. SequoiaDB using PCRE regex.



Note: \$regex and \$options Supporting the use of.

## \$options

\$options Offers four select flay:

- i:Setting this modifier mode for case-insensitive match letters.
- m:By default, pcre that the target string is composed by a single character, "the line" metacharacter (^) matches only at the beginning of the string, and the "end of line" metacharacter (\$) matches only at the end of the string, or the last newline. When this modifier is set, the "first line" and "end of line" will match the target string before or after any newline, In addition, most were matching the target string beginning and the very end position, if the target string no "W n", or pattern does not appear ^ or \$, setting this modifier does not have any effect.



- x:Setting this modifier mode without escaped or is not blank character class data characters will always be ignored, and is located in an unescaped # character outside a character class and the next newline characters between lines have been ignored.
- s:Setting this modifier mode dot metacharacter matches all characters, including newlines, without this modifier, the point number does not match a newline.

example

- Back to collection bar next str field values match case-insensitive regular expressions dh.\*fj records  
`db.foo.bar.find({str:{$regex:'dh.*fj',$options:'i'}})`

## Update Operator

update operator	description	information	samples
<code>\$inc</code>	increase	add the given value to the value of specified field	<code>db.foo.bar.update({\$inc:{age:25}})</code>
<code>\$set</code>	set the specified field	set the given value to the specified field	<code>db.foo.bar.update({\$set:{age:10}})</code>
<code>\$unset</code>	delete the specified field	delete the specified field of object	<code>db.foo.bar.update({\$unset:{age:''}})</code>
<code>\$addto</code>	add to set	if the given element doesn't exist in the array,then added,otherwise skipped.the object field must be array type	<code>db.foo.bar.update({\$addto:{array:[3,4,5]}})</code>
<code>\$pop</code>	pull the last of N values in the array	pull the last of N values in the array,the object field must be array type(if N less than 0 means that delete the first of -N values from the beginning of array).	<code>db.foo.bar.update({\$pop:{array:2}})</code>
<code>\$pull</code>	pull values	pull the specified value from the object array , the object element must be array.	<code>db.foo.bar.update({\$pull:{array:2}})</code>
<code>\$pull_all</code>	pull array	pull each of the elements in the specified array from the object array,the object element must be array type.	<code>db.foo.bar.update({\$pull_all:{array:[2,3,4]}})</code>
<code>\$push</code>	push values	push the value to the object array,the object element must be array type.	<code>db.foo.bar.update({\$push:{array:2}})</code>
<code>\$push_all</code>	push array	push each of values in the specified array to the object array,the object element must be array type.	<code>db.foo.bar.update({\$push_all:{array:[2,3,4]}})</code>

`$inc`

Grammar

`{ $inc: { <field name1>: <value1>, <field name2>: <value2>, ... } }`

Description

`$inc` operation is to increase the specified "<value>" for the specified "<field name>".If these is no specified field name,then add the field name and value to the record; If the specified field name exist in records, then the value of the specified field name will plus the specified value.

## Sample

- select records in the collection bar where the field name age has the value greater than 15, then update them, the value of field name age will increase 5, and the value of field name ID will increase 1.

```
db.foo.bar.update({$inc: {age: 5, ID: 1}}, {age: {$gt: 15}})
```

- select records in the collection bar where the field name arr existed and is array type, and set the value of second element to increase 1.

```
db.foo.bar.update({$inc: {"arr.1": 1}}, {arr: {$exists: 1}})
```

## \$set

## Grammar

```
{$set:<{<field name1>:<value1>,<field name2>:<value2>,...}>}
```

## Description

\$set is used to update the value of specified field name (field name1, field name2) to the specified value (value1, value2). If there is no the specified field name in the records, then fill the field name and value to the record, else, update the value to the specified value.

## Samples

- Select records which the field age is not exist, and use '\$set' update those records.

```
db.foo.bar.update({$set: {age: 5, ID: 10}}, {age: {$exists: 0}})
```

- Update all the records in the collection bar, set the value of field str to "abc"

```
db.foo.bar.update({$set: {str: "abc"}})
```

- Using '\$set' update the element of nested array object. The field arr is a nested array object in collection bar, for example, there have two records: {arr:[1,2,3], name:"Tom"}, {name:"Mike", age:20}, and the second record doesn't exist arr field.

```
db.foo.bar.update({$set: {"arr.1": 4}}, {name: {$exists: 1}})
```

This operation is to select records which contain name field, then use '\$set' to update the array object arr. If there is no array object arr in original record, then \$set will add the arr field as a array object to the record. The above two records will update as:

```
{arr: [1, 4, 3], name: "Tom"}, {arr: {"1": 4}, name: "Mike", age: 20}
```

## \$unset

## Grammar

```
{$unset:<{<field name1>:"", <field name2>:"", ...}>}
```

## Description

\$unset operation is used to delete the specified field name. If the specified field does not exist in record, then skipped.

## Samples

- Delete the fields name and age of records in the collection bar, if a record does not contain the field name or age, then skipped, do nothing.

```
db.foo.bar.update({$unset: {name: "", age: ""}})
```

- Use '\$unset' to delete the element of array object. There has a record for example: {arr: [1,2,3], name: "Tom"}. The operation use \$unset to delete the second element as follows:

```
db.foo.bar.update ( {$unset: {"arr. 2": ""}} )
```

after this operation, the record update as: {arr: [1, null, 3], name: "Tom"}

- Use \$unset to delete the element of nested object. There has a record for example: {content: {ID: 1, type: "system", position: "manager"}, name: "Tom"},

content is a nested object, it has two elements: ID and type. The operation use \$unset to delete element type as follows:

```
db.foo.bar.update ( {$unset: {"content. type": ""}} )
```

after this operation, the record update as: {content: {ID: 1, position: "manager"}, name: "Tom"}

## \$addto

### Grammar

```
{ $addto: { <field name1>: [ <value1>, <value2>, ..., <valueN> ] , <field name2>: [ <value1>, <value2>, ..., <valueN> ], ... } }
```

### Description

\$addto is to add element and value to the array object, the operation object must be array type field. \$addto has the following rule:

- the records have the specified fields (<field name1>, <field name2> ...).  
if the specified values ([<value1>, <value2>, ..., <valueN>]) exist in the records, skipping with no operations, only add values that doesn't exist to the target array.
- the records doesn't exist the specified field name.

If the records itself doesn't exist the specified field name (<field name1>, <field name2> ...), then add the specified field names and values to the records.

### Sample

- the records exist in the target array object. As the following record: {arr: [1, 2, 4], age: 10, name: "Tom"}

```
db.foo.bar.update ( { $addto: { arr: [1, 3, 5] } }, { arr: { $exists: 1 } } )
```

after this operation, the record update as : {arr: [1, 2, 4, 3, 5], age: 10, name: "Tom"}. in the previous record, there is no 3 or 5 element in the array object arr, after using \$addto operator, update them to arr.

- the records don't exist in the specified array object, as the following record: {name: "Mike", age: 12}

```
db.foo.bar.update ( { $addto: { arr: [1, 3, 5] } }, { arr: { $exists: 0 } } )
```

after this operation, the record update as: {arr: [1, 3, 5], age: 12, name: "Mike"}. in the previous record, there is no array object arr, after using \$addto operator, update the array object arr and its elements to the record.

## \$pop

### Grammar

```
{ $pop: { <field name1>: <N>, <field name2>: <N>, ... } }
```

### Description

\$pop operation is to delete the last of N elements in the array object(<field name1>,<field name2>,...), the operation object must be array type.If the records don't exist the specified array object,skipping with no operations;if N is greater than the length of array object,then the length of array object update to zero,that is all of the element will drop;if N is less than zero,means to delete the first of -N elements from the beginning of array object.

### Sample

- delete the last two of elements from the array object arr in the collection bar.As the following record:  
{arr:[1,2,3,4],age:20,name:"Tom"}

```
db.foo.bar.update({$pop: {arr: 2}})
```

after this operation,the record update as:{arr:[1,2],age:20,name:"Tom"}

- delete the last ten of elements from the array object arr in the collection bar.as the following record:  
{arr:[1,2,3,4],age:20,name:"Tom"}

```
db.foo.bar.update({$pop: {arr: 10}})
```

after this operation,the record update as:{arr:[],age:20,name:"Tom"}

- delete the first two of elements from the array object arr in the collection bar,that is set N to -2.as the following record:{arr:[1,2,3,4],age:20,name:"Tom"}

```
db.foo.bar.update({$pop: {arr: -2}})
```

after this operation,the record update as:{arr:[3,4],age:20,name:"Tom"}

### \$pull

### Grammar

```
{ $pull: {<field name1>:<value1>,<field name2>:<value2>,...}}
```

### Description

\$pull operation is to delete the specified values(<value1>,<field name2>:<value2>,...) from the given array object field(<field name1>,<field name2>), the operation object must be array type field. If the specified array object don't exist in record,skipping with no operations; If the specified values don't exist in array object, also skipping with no operations.

### Sample

- delete the elements which have the value 2 from the array object arr and have the value "Tom" from array object name in the collection bar, as the following record:{arr[1,2,4,5],age:10,name:["Tom","Mike"]}

```
db.foo.bar.update({$pull: {arr: 2, name: "Tom"}})
```

after this operation,the record update as:{arr[1,4,5],age:10,name:["Mike"]}

- delete the elements which have the value 2 from the array object arr and have the value "Tom" from array object name in the collection bar, as the following record:{arr[1,3,4,5],age:10,name:["Tom","Mike"]}

```
db.foo.bar.update({$pull: {arr: 2, name: "Tom"}})
```

after this operation,the record update as:{arr[1,3,4,5],age:10,name:["Mike"]}. As there is no element which has the value 2 in the array object arr ,so object arr have no change.

## \$pull\_all

### Grammar

```
{ $pull_all: { <field name1>: [ <value1>, <value2>, ..., <valueN> ], <field name2>: [ <value1>, <value2>, ..., <valueN> ], ... } }
```

### Description

\$pull\_all is to pull the given values([<value1>, <value2>, ..., <valueN>]) of the specified array object(<field name1>), the operation object must be array type field.

if the specified array object does not exist in the records, skipping with no operations; if the specified values don't exist in array object, nor do any operation.

### Sample

- delete the elements which have the value 2 or 3 from the array object arr and have the value "Tom" from array object name in the collection bar, as the following record: {arr:[1,2,4,5],age:10,name:["Tom","Mike"]}

```
db.foo.bar.update ( { $pull_all: { arr: [ 2, 3 ], name: [ "Tom" ] } } )
```

after this operation, the record update as: {arr:[1,4,5],age:10,name:["Mike"]}

- delete the elements which have the value 4 or 5 from array object arr in the collection bar, as the following record: {arr:[1,3,4,5],age:10,name:["Tom","Mike"]}

```
db.foo.bar.update ( { $pull_all: { arr: [ 4, 5 ] } } )
```

after this operation, the record update as: {arr:[1,3],age:10,name:["Tom","Mike"]}.

## \$push

### grammar

```
{ $push: { <field name1>: <value1>, <field name2>: <value2>, ... } }
```

### description

\$push A value of (<value1>) is inserted into the given destination array (<value1>), Operand must be an array type field. If the record does not exist in the specified field name, field names will be specified in the form of an array object pushed into the records and populate its specified value; if the record exists in the specified field name and field names specified value exists, specify the value will be pushed into the record.

### example

- The collection bar under arr array object push the value 1. Original records exist elements of an array object arr, if records: {arr:[1,2,4],age:10,name:["Tom","Mike"]}

```
db.foo.bar.update ( { $push: { arr: 1 } } )
```

After this operation, record updates: {arr [1,2,4,1], age: 10, name: ["Tom", "Mike"]}. Although there are elements of the original arr 1, using \$push operator, or adds an element a push to arr array object.

- Push bar to the collection does not exist in an array of objects and values. Record does not exist in the original array object name, if any, records: {arr:[1,2],age:20}

```
db.foo.bar.update ( { $push: { name: "Tom" } }, { name: { $exists: 0 } } )
```

After this operation, record updates: {arr [1,2], age: 20, name: ["Tom"]}. Record does not exist in the original array object name, use \$push operator will name an object in the form of an array pushed into the record.

\$push\_all

Grammar

```
{ $push_all: { <field name1>: [ <value1>, <value2>, ..., <valueN> ], <field2>: [ <value1>, <value2>, ..., <valueN> ], ... } }
```

Description

\$push\_all is used to push each of the specified values ([<value1>, <value2>, ..., <valueN>]) to the specified array object. the operation object must be array type field. if the specified array object does not exist in the records, then push the array object and all the values ([<value1>, <value2>, ..., <valueN>]); if the specified values exist in array object, push them to the specified array object as the same.

Samples

- Push array [1,2,8,9] to the array object arr in the collection. There is a record like this: {arr[1,2,4,5], age:10, name:["Tom", "Mike"]}

```
db.foo.bar.update ( { $push_all: { arr: [ 1, 2, 8, 9 ] } } )
```

after this operation, the record update as: {arr[1,2,4,5,1,2,8,9], age:10, name:["Mike"]}, although, the arr object has elements 1 and 2 in original record, using \$push\_all will push all the elements of array [1,2,8,9] to array object arr.

- Push array object name to collection bar, assuming the original record does not exist array object name, there is a record like this: {arr[1,3,4,5], age:10}

```
db.foo.bar.update ( { $push_all: { name: [ "Tom", "Jhon" ] } }, { name: { $exists: 0 } } )
```

after this operation, the record update as: {arr[1,3,4,5], age:10, name:["Tom", "Mike"]}.

## Aggregate Operator

Parameter name	Description	Example
<a href="#">\$project</a>	Select the field name to be output, "1" indicates that the output "0" means no output, you can also implement fields.	{ \$project: { field1: 1, field0: 0, alias: "\$field3" } }
<a href="#">\$match</a>	Select the matching criteria to achieve from a collection of records, quite where the SQL statement.	{ \$match: { field: { \$lte: value } } }
<a href="#">\$limit</a>	Limit the number of records returned.	{ \$limit: 10 }
<a href="#">\$skip</a>	Start point control of the result set, the result set that skips the number of records specified.	{ \$skip: 5 }
<a href="#">\$group</a>	Achieve grouping of records, similar to SQL's group by statement, "_id" designated group field.	{ \$group: { _id: "\$field" } }
<a href="#">\$sort</a>	Achieve the sort of result set "1" represents Ascending, "-1" for descending.	{ \$sort: { field1: 1, field2: -1, ... } }

\$group Aggregation operator supports the following aggregate functions:

Function name	Description
<a href="#">\$addToSet</a>	Adds the specified field values in the array, the same field value will be added once.

Function name	Description
<a href="#">\$first</a>	Take packet field values in the first record.
<a href="#">\$last</a>	Take packet field value in the record.
<a href="#">\$max</a>	Take the maximum value of the packet field.
<a href="#">\$min</a>	Take the smallest packet field values.
<a href="#">\$avg</a>	Take the average packet field values.
<a href="#">\$push</a>	Add all the fields to the array, even if the same array of field values that already exist, and continue to add.
<a href="#">\$sum</a>	Take the sum of packet field values.

## \$project

### Description

\$project similar to the SQL select statement, by using \$project operations can filter out the required fields from the record, the value is 1 if the field name, which means that selected to 0 means no selection, can also be achieved rename fields.



Note: If the record does not exist for the selected field, then the following output format: "field":null, field to field name does not exist. Nested object using the dot operator(.)

### Example

- Use \$project quickly select from a focus on the results of field Related information required.

```
db.collection.space.collection.aggregate({ $project : {title: 0, author: 1}})
```

This operation is selected author field, while title field in the result set is not output.

- Use \$project rename the field name, as follows:

```
db.collection.space.collection.aggregate({ $project : {author: 1, name: "$studentName", dep: "$info.department"}})
```

This operation will rename the field names studentName name to output, the info object in the field of child objects department rename dep. Nested object field references using the dot operator (.) Point.

- The following example uses \$project to select the output field, then use [\\$match](#) conditional matching records.

```
db.collection.space.collection.aggregate({ $project: {score: 1, author: 1}}, { $match: {score: { $gt: 80 } } })
```

This output using \$project and author of all score field values, and then press the \$match record output matching conditions.



Note: Due \$project selected output field names, field names so [\\$match](#) \$project must be selected field name.

## \$match

### Description

\$match with find() method parameters are identical cond, \$match can be achieved by matching criteria to select records from the collection.

\$match grammar rules, please refer to the read operation [find\(\)](#) cond parameter method introduced.

### Example

- The following example uses \$match perform simple matching

```
db.collection.space.collection.aggregate({ $match : { $and: [ {score: 80}, {"info.name": {$exists: 1}} ] } })
```

This operation represents the return qualifying score from a set collection of =80 and the info object child object name field is present record.

- The following example uses \$match matching records that meet that criteria, then the result set using \$group grouping final output using \$project focused to the specified field name.

```
db.collection.space.collection.aggregate({ $match : { $and: [ {score: 80}, {"info.name": {$exists: 1}} ] } }, {$group: {_id: "$major"}}, {$project: {major: 1, dep: 1}})
```

The first set of collection operations in return qualifying score=80 and recording info objects child objects exist in the name field, and then grouped according to major fields, the final choice of major, and dep field output in the result set.

### \$group

#### Description

\$group grouping to achieve the result set, similar to the group by SQL statement. First, specify the grouping key(\_id), by "\_id" to identify the grouping field, the field can be a single packet, it can be more than one, the format is as follows:

Single packet key: {\_id: "\$field"}

Multiple grouping key: {\_id: {field1: "\$field1", field2: "\$field2", ...}}



**Note:** Use \$group must specify the \_id field, when \_id is null, ie {\_id:null}, means no grouping.

Nested object using the dot operator (.) Reference field names.

### Example

- \$group using the following:

```
db.collection.space.collection.aggregate({ $group: {_id: "$major", avg_score: {$avg: "$score", Major: {$first: "$major"}} } })
```

This operation reads the records from the collection represents the collection, grouped according to major fields. In the result set, taking the first record of each grouping of major fields, rename Major, field to score the value of each packet average, rename ave\_score. Returns are as follows:

```
{
  "avg_score": 82,
  "major": "optics"
}
{
  "avg_score": 77.25,
  "major": "physics"
}
```

\$group Aggregation operator supports the following aggregate functions:

Function name	Description
<a href="#">\$addToSet</a>	Adds the specified field values in the array, the same field value will be added once.
<a href="#">\$first</a>	Take packet field values in the first record.
<a href="#">\$last</a>	Take packet field value in the record.
<a href="#">\$max</a>	Take the maximum value of the packet field.
<a href="#">\$min</a>	Take the smallest packet field values.



Function name	Description
<code>\$avg</code>	Take the average packet field values.
<code>\$push</code>	Add all the fields to the array, even if the same array of field values that already exist, and continue to add.
<code>\$sum</code>	Take the sum of packet field values.

## `$addtoiset`

### Description

After recording grouping, use `$addtoiset` the specified field values added to the array, the same field value will be added once. Nested object using the dot operator(.) Reference field names.

### Example

- Follows the specified field values recorded packet will add to the array output

```
db.collectionspace.collection.aggregate({$group: {_id: "$dep", dep:
{$first: "$dep"}, addtoiset_major: {$addtoiset: "$major"}}})
```

This operation records are grouped by `dep` field values and use the `$first` output of the first record in each group `dep` field, major field values turn into an array using `$addtoiset` returned, the output field called `addtoiset_major`, as follows:

```
{
  "Dep": "Institute of Physics and Electronics",
  "addtoiset_major": [
    "Physics",
    "Optics",
    "Electricity"
  ]
}
{
  "Dep": "Computer Academy",
  "addtoiset_major": [
    "Computer Science and Technology",
    "Computer Software and Theory",
    "Computer Engineering"
  ]
}
```

## `$first`

### Description

After recording group, take a packet field values specified in the first record of the nested object using the dot operator(.) Reference field names.

### Example

- After recording the value of the specified field of the packet, the output of the first record in each group.

```
db.collectionspace.collection.aggregate({$group: {_id: "$dep", Dep: {$first: "$dep"}, Name:
{$first: "$info.name"}}})
```

This grouping records by `dep` field, take each packet `dep` first record field values and nested object `name` field value, the output field names were `Dep` and `Name`, records are returned as follows:

```
{
  "Dep": "School of Physics and Electronics",
  "Name": "Lily"
}
```

```
{
  "Dep": "Computer Academy",
  "Name": "Tom"
}
```

\$last

### Description

After recording grouping, grouping taken last record specified field values, nested object using the dot operator(.) Reference field names.

### Example

- After recording the value of the specified field grouping, the output of each group last record.

```
db.collection.space.collection.aggregate({$group: {_id: "$dep", Major:
{$addtoSet: "$major"}, Name: {$last: "$info.name"}}})
```

This grouping records by dep field, using \$last taken the last record in each grouping nested object name field value, the output field called Name, and the major field value in each packet using \$addtoSet fill the array Back to return to the field named Major, records are returned as follows:

```
{
  "Major": [
    "Physics",
    "Optics",
    "Electricity"
  ],
  "Name": "Kate"
}
{
  "Major": [
    "Computer Sciecne and Technology",
    "Computer Software and Theory",
    "Computer Engineering"
  ],
  "Name": "Jim"
}
```

\$max

### Description

After recording the maximum packet, take pakcet return the specified field of nested objects using the dot operator(.) Reference field names.

### Example

- After the recording of the group, return the maximum packet specified field.

```
db.collection.space.collection.aggregate({$group: {_id: "$dep", max_score: {$max: "$score"}, Name:
{$last: "$info.name"}}})
```

This grouping records by dep field, use \$max returns the maximum value in each packet, the output field score a field named max\_score, and use \$last taken the last record in each group nested object name field value, the output field called name. Records are returned as follows:

```
{
  "max_score": 93,
  "Name": "Kate"
}
{
  "max_score": 90,
  "Name": "Jim"
}
```

```
}
```

## \$min

### Description

After recording the minimum packet, take packet returns the specified field of nested objects using the dot operator(.) Reference field names.

### Example

- After the recording of the group, return the minimum packet specified field.

```
db.collection.space.collection.aggregate({$group: {_id: "$dep", min_score: {$min: "$score"}, Name: {$last: "$info.name"}}})
```

This grouping records by dep field, use \$min returns the minimum value in each packet, the output field score a field named in\_score, and use \$last taken the last record in each group nested object name field value, the output field called name. Records are returned as follows:

```
{
  "min_score": 72,
  "Name": "Kate"
}
{
  "min_score": 69,
  "Name": "Jim"
}
```

## \$avg

### Description

After recording the average grouping, take the packet returns the specified field of nested objects using the dot operator(.) Reference field names.

### Example

- After the recording of the group, returns the average packet specified field.

```
db.collection.space.collection.aggregate({$group: {_id: "$dep", avg_age: {$avg: "$info.age"}, max_age: {$max: "$info.age"}, min_age: {$min: "$info.age"}}})
```

This grouping records by dep field, use \$avg returns the average of the output fields for each age group nested object field named avg\_age, then use \$min returns the minimum age for each grouping field nested objects the output field named min\_age, use \$max returns the maximum value in each packet nesting, the output field named field object age max\_age. Records are returned as follows:

```
{
  "avg_age": 23.727273,
  "max_age": 36,
  "min_age": 15
}
{
  "avg_age": 24.5,
  "max_age": 30,
  "min_age": 20
}
```

## \$sum

### Description

after recording group, returns the sum of each group in the specified field values for nested objects using the dot operator(.) Reference field names.

### Example

- After recording the sum of the packet, the packet returns the specified field values.

```
db.collection.space.collection.aggregate({$group: {_id: "$dep", sum_score: {$sum: "$score"}, Dep: {$first: "$dep"}}})
```

This sum dep field grouping records by using \$sum return score in each grouping field value, the output field named sum\_score, then use [\\$first](#) take each packet dep first record field value, the output field name to Dep. Records are returned as follows:

```
{
  "sum_score": 888,
  "Dep": "Electrospray Academy"
}
{
  "sum_score": 476,
  "Dep": "Computer Academy"
}
```

## \$push

### Description

After recording grouping, use \$push Adds the specified field values to the array, even if the same value in the array already exists, continue to add. Nested object using the dot operator (.) Reference field names.

### Example

- Follows the specified field values recorded packet will add to the array output.

```
db.collection.space.collection.aggregate({$group: {_id: "$dep", Dep: {$first: "$dep"}, push_age: {$push: "$info.age"}}})
```

This operation dep field values recorded by grouping values for each age group in the field of nested objects into an array using \$push returned, the output field named push\_age, as follows:

```
{
  "Dep": "School of Physics and Electronics",
  "push_age": [
    28,
    18,
    20,
    30,
    28,
    20
  ]
}
{
  "Dep": "Computer Academy",
  "push_age": [
    25,
    20,
    22
  ]
}
```

## \$limit

### Description

\$limitAchieved in the result set returned by limiting the number of records. If the specified number of records is greater than the actual total number of records, then return the actual total number of records.

### Example

- Limit the result set returned the first 10 records.

```
db.collectionspace.collection.aggregate( { $limit : 10 } )
```

This operation represents a collection of collection to read the first 10 records.

## \$sort

### Description

\$sort Is used to specify the collation of the result set. Nested object using the dot operator(.) Reference field names.

### Example

```
db.collectionspace.collection.aggregate( {$sort: {score: -1, name: 1}} );
```

This operation represents read records from the collection collection in the field and score values were sorted in descending order (1 indicates ascendign, -1 means descending).

When the score between the record field values are the same, then the name field values in ascending order.

## \$skip

### Description

\$skip Parameter controls the starting point of the result set, the result set that is skipping the number of records specified. If the number of records is greater than the total number of records to skip, returns 0 records.

### Example

- Skip 10 records returned.

```
db.collectionspace.collection.aggregate( { $skip : 10 } ) ;
```

This operation represents a collection of collection of records read from and skip ahead 10, began to return from the first 11 records.

## Mapping Table fromSQL to Aggregate

The following table describes the main SQL keywords SequoiaDB aggregation operator table.

SQL keywords	SequoiaDB aggregation operator
where	\$match
group by	\$group
having	\$match
select	\$project
order by	\$sort
top	\$limit
offset	\$skip

SQL keywords	SequoiaDB aggregation operator

The following table describes the standard SQL statements and control statements SequoiaDB gathered between.

SQL statement	SequoiaDB gather statement	Description
select product_id as p_id, price from table	db.cs.table.aggregate({\$project: {p_id: "\$product_id", price: 1, date: 0}})	Return product_id and price fields for all records where product_id rename p_id, on the record date field is not returned.
select sum(price) as total from table	db.cs.table.aggregate({\$group: {_id: null, total: {\$sum: "\$price"}}})	Price for table summing the values in the field, and rename total.
select product_id, sum(price) as total from table group by product_id	db.cs.table.aggregate({\$group: {_id: "\$product_id", product_id: {\$first: "\$product_id"}, total: {\$sum: "\$price"}}})	Table of records grouped by product_id field. Find field in each packet accumulation and price value, and rename total.
select product_id, sum(price) as total from table group by product_id order by total	db.cs.table.aggregate({\$group: {_id: "\$product_id", product_id: {\$first: "\$product_id"}, total: {\$sum: "\$price"}}, {\$sort: {total: 1}})	Table of records grouped by product_id field. Find the value of the field in each packet price and accumulated, and rename total. The result set by the total value of the field names in ascending order.
select product_type_id, product_id, sum(price) as total from table group by product_type_id, product_id	db.cs.table.aggregate({\$group: {_id: {product_type_id: "\$product_type_id", product_id: "\$product_id"}, product_type_id: {\$first: "\$product_id"}, total: {\$sum: "\$price"}}})	Table records in a field by first grouping by product_type_id, then product_id field grouping. Find the value of the field in each packet price and accumulated, and rename total.
select product_id, sum(price) as total from table group by product_id having total > 1000	db.cs.table.aggregate({\$group: {_id: "\$product_id", product_id: {\$first: "\$product_id"}, total: {\$sum: "\$price"}}, {\$match: {total: {\$gt: 1000}}})	Table of records grouped by product_id field. Find the value of the field in each packet price and accumulated, and rename total. Total return only to meet the conditions grouped field values greater than 1000.
select product_id, sum(price) as total from table where product_type_id = 1001 group by product_id	db.cs.table.aggregate({\$match: {product_type_id: 1001}}, {\$group: {_id: "\$product_id", product_id: {\$first: "\$product_id"}, total: {\$sum: "\$price"}}})	Select product_type_id = 1001 qualifying record. For the selected records are grouped by product_id. The price for each packet field values on and, and rename total.
select product_id, sum(price) as total from table where product_type_id = 1001 group by product_id having total > 1000	db.cs.table.aggregate({\$match: {product_type_id: 1001}}, {\$group: {_id: "\$product_id", product_id: {\$first: "\$product_id"}, total: {\$sum: "\$price"}}, {\$match: {total: {\$gt: 1000}}})	Select product_type_id = 1001 qualifying record. For the selected records are grouped by product_id. The price for each packet field values on and, and rename total. Total return only to meet the conditions grouped field values greater than 1000.
select top 10 * from table	db.cs.table.aggregate({\$group: {_id: null}}, {\$limit: 10})	Returns the result set before 10 records.
select * from table offset 50 rows fetch next 10	db.cs.table.aggregate({\$group: {_id: null}}, {\$skip: 50}, {\$limit: 10})	After skipping the results of 50 focus before recording, return the next 10 records.

## SQL Grammar

Sequoiadb is a document-oriented type of non-relational database, this section mainly describes how to use SQL to access and process data in the Sequoiadb system.



Note: in the sequoiaDB, SQL statement is not case sensitive.

### SQL grammar table

statement	description	samples
<a href="#">create collectionspace</a>	create collectionspace	db.execUpdate("create collectionspace foo")
<a href="#">drop collectionspace</a>	drop collectionspace	db.execUpdate("drop collectionspace foo")
<a href="#">create collection</a>	create collection	db.execUpdate("create collection foo.bar")
<a href="#">drop collection</a>	drop collection	db.execUpdate("drop collection foo.bar")
<a href="#">create index</a>	create index	db.execUpdate("create index test_index on foo.bar (age)")

statement	description	samples
<a href="#">drop index</a>	drop index	db.execUpdate("drop index test_index on foo.bar")
<a href="#">list collectionspace</a>	list collectionspace	db.exec("list collectionspace")
<a href="#">list collections</a>	list collection	db.exec("list collections")
<a href="#">insert into</a>	insert	db.execUpdate("insert into foo.bar(age,name) values(20,W"TomW")")
<a href="#">select from</a>	query	db.exec("select * from foo.bar")
<a href="#">update set</a>	update	db.execUpdate("update foo.bar set age=25")
<a href="#">delete from</a>	delete	db.execUpdate("delete from foo.bar")
<a href="#">group by</a>	group	db.exec("select dept_no,count(emp_no) as Number of employees from foo.bar group by dept_no ")
<a href="#">order by</a>	sort	db.exec("select * from foo.bar order by age desc")
<a href="#">split by</a>	records split	
<a href="#">limit</a>	limit the number of return records	db.exec("select * from foo.bar limit 5")
<a href="#">offset</a>	set the number of records skipped	db.exec("select * from foo.bar offset 5")
<a href="#">as</a>	aliases	
<a href="#">join on</a>	join	
<a href="#">left outer join on</a>	left join	
<a href="#">right outer join on</a>	right join	
<a href="#">count</a>	count	
<a href="#">sum</a>	sum	
<a href="#">avg</a>	average	
<a href="#">max</a>	max	
<a href="#">min</a>	min	
<a href="#">first</a>	select the first data	
<a href="#">last</a>	select the last one data	
<a href="#">push</a>	combinde array	
<a href="#">addtoaset</a>	no duplicate values in the array merge	
<a href="#">buildobj</a>	merging objects	
<a href="#">mergearrayset</a>	merger does not contain duplicate array of fields	

## sql create collectionspace

create collectionspace

Be used to create collectionspace in the database .

Grammar

```
create collectionspace <cs_name>
```

<cs\_name> : collectionspace name. the length of collectionspace can't over 128Byte,and collectionspace name can't be empty.

## Sample

- create collectionspace foo

```
db.execUpdate("create collectionspace foo") //Equivalent db.createCS("foo")
```

## sql drop collectionspace

## drop collectionspace

Be used to drop the collectionspace in the database.

## Grammar

```
drop collectionspace <cs_name>
```

<cs\_name>collectionspace name,the collectionspace name must be exist in the database.

## Sample

- This sample will drop the collectionspace foo.

```
db.execUpdate("drop collectionspace foo") //Equivalent db.dropCS("foo")
```

## sql create collection

## create collection

Be used to create collectio, must specify the collection in which collectionspace.

## Grammer

```
create collection <cs_name>.<cl_name>
```

<cs\_name> : collectionspace in database

<cl\_name> : collection name.the length of it can't exceed 127Byte,and collection name can't be empty ,and can't exist same collection name in a collectionspace.

## Sample

- create collection bar in the collectionspace foo

```
db.execUpdate("create collection foo.bar") //Equivalent db.foo.createCL("bar")
```

## sql drop collection

## drop collection

Be used to drop the collection in the collectionspace.

## Grammar

```
drop collection <cs_name>.<cl_name>
```

<cs\_name>collectionspace name in the database.the collectionsapce name must be in the database.

<cl\_name>collection name. collection name also must be in the collectionsapce.



### Sample

- The following sample will drop the collection bar in the collectionspace foo.

```
db.execUpdate("drop collection foo.bar") //Equivalent db.foo.dropCL("bar")
```

## sql create index

### create index

Be used to create index in the collections.in the case of without reading collection,indexes enable database application to find data faster.

### Grammer

```
create [unique] index <index_name> on <cs_name>.<cl_name> (field1_name [asc|desc],...)
```

[unique] : TO logo if the created index is unique.unique index means that the same field name can't have the same index name.

<index\_name> : index name

<cs\_name> : collectionspace name

<cl\_name> : collection name

field1\_name : field name where create index , with an index name can be created on multiple fields.

[asc|desc] : sort, asc stand for ascending index the value of a field.desc stand for descending index the value of a field.the default is asc.

### Sample

- This sample will create a simple index, the name is "test\_index" on the field name age in the collectionspace foo,collection bar.

```
db.execUpdate("create index test_name on foo.bar (age) ")
```

if want to descending index the value of a field,can add the reserved word desc after the field name.

```
db.execUpdate("create index test_name on foo.bar (age desc) ")
```

if want to the index in more than one field ,can list all the field name in the brackets,and separated by commas.

```
db.execUpdate("create index test_name on foo.bar (age desc,name asc) ")
```

- the following sample will create a unique index on the field age.

```
db.execUpdate("create unique index test_name on foo.bar (age) ")
```

## sql drop index

### drop index

Be used to drop the index in the collection.

### Grammar

```
drop index <index_name> on <cs_name>.<cl_name>
```

<index\_name> : index name

<cs\_name> : collectionspace name

<cl\_name> : collection name

## Sample

- The following sample will drop the index name "test\_index" in the collectionspace foo, collection bar. Assume the index name is already exist.

```
db.executeUpdate("drop index test_index on foo.bar") //Equivalent
db.foo.bar.dropIndex("test_index")
```

## sql list collectionspaces

## list collectionspaces

Be used to list all the collectionspaces in the database.

## Grammar

```
list collectionspaces
```

## Sample

- This sample will return all the collectionspaces in the database.

```
db.exec("list collectionspaces")
result:
{
  "Name": "testfoo"
  "Name": "big"
  ...
}
```

## sql list collections

## list collections

Be used to list all the collections in the collectionspace.

## Grammar

```
list collections
```

## Sample

- This sample will return all the collections.

```
db.exec("list collectionspaces")
result:
{
  "Name": "testfoo.testbar"
  "Name": "big.small"
  ...
}
```

## sql insert into

## insert into

Be used to insert new records to the collection.

## Grammar

```
insert into <cs-name>.<cl-name>(<field1-name,field2-name,...>) values(<value1,value2,...>)
or
```

```
insert into <cs_name>.<cl_name> <select_set>
```

<cs\_name> : collectionspace name

<cl\_name> : collection name

<field\_name> : field name

<value> : values the field name corresponding

<select\_set> : query result set

### Sample

- This sample will insert one record to the collection bar,field names are age and name,corresponding values are 25 and "Tom".

```
db.execUpdate("insert into foo.bar(age,name) vaules (25,\"Tom\") ")
```

- This sample will insert batch records to the collection bar,these records is the result set select from collection small.

```
db.execUpdate("insert into foo.bar select * from big.small")
```

## sql select

select

Be used to select data from collection,the result will be storage as a result set.

### Grammar

```
select * from <cs_name>.<cl_name>
or
select <field1_name,field2_name,...> from <cs_name>.<cl_name>
```

<cs\_name> : collectionspace name

<cl\_name> : collection name

<field\_name> : field name

### Sample

- This sample will return the select field name.if a match record doesnot contain the specified field name,then return null value .

```
db.exec("select age,name from foo.bar")
result:
{
  "age": 10,
  "name": null
}
{
  "age": 10,
  "name": "Tom"
}
...
```

- This sample will return all records of all the field name .

```
db.exec("select * from foo.bar")
result:
{
  "_id": {
    "$oid": "51c909b0c5b855e029000000"
  },

```

```

    "age": 10
  },
  {
    "_id": {
      "$oid": "51c909b9c5b855e029000001"
    },
    "age": 10,
    "name": "Tom"
  },
  {
    "_id": {
      "$oid": "51c909c2c5b855e029000002"
    },
    "age": 10,
    "name": "Tom",
    "phone": 123456
  }
}...

```

**Note:**

1. you can select like [where](#)、[group by](#)、[order by](#)、[limit](#)、[offset](#) keywords to select those records you want.
2. if the query source is not collection, then all the field name in this layer query need to use alias (\* except)  
example: select T.a , T.b from (select \* from foo.bar) as T where T.a<10
3. subquery must use alias. the alias appear in subquery only act on upper layer.

## sql update

### update

Be used to modify records in the collection.

### Grammar

```
update <cs_name>.<cl_name> set (<field1_name>=<value1>,...) [where <condition>]
```

<cs\_name> : collectionspace name

<cl\_name> : collection name

<condition> : condition, only update the records match the condition.

### Sample

- This sample will modify all the records in the collection, the value of field name age will be update 20, if a records doesnot contain feild age, then 'age:20' will be add to the record.

```
db.executeUpdate("update foo.bar set age=20")
```

- This sample will be modify the match records, only update the records that satisfy the condition age<10.

```
db.executeUpdate("update foo.bar set age=20 where age<10")
```

## sql delete

### delete

Be used to delete records in the collection.

## Grammar

```
delete from <cs_name>.<cl_name> [where <condition>]
```

<cs\_name> : collectionspace name

<cl\_name> : collection name

<condition> : condition,only delete the records which match the condition.

## Sample

- This sample will delete all the records in the collection.

```
db.executeUpdate("delete from foo.bar")
```

- This sample will delete the records which match the condition age<10.

```
db.executeUpdate("delete from foo.bar where age<10")
```

## sql group by

### group by

Be used to group result set According to one or more filed name combined with aggragate function.

## Grammar

```
group by <field1_name [ASC|DESC ], ...>
```

<field\_name> : field name

[asc|desc] : sort, asc stand for ascending,desc stand for descending. default asc.

## Sample

- Calculating the number of employees in each department and group by the field name dept\_no.

```
db.exec("select dept_no, count(emp_no) as 员工总数 from foo.bar group by dept_no")
```



Note:

likesum , count , min , max , avgthese counting function must use alias.

## sql order by

### order by

Be used to sort the result set according to the specified field name,default asc.

## Grammar

```
order by <field1_name [ASC|DESC ], ...>
```

<field\_name> : field name

[asc|desc] : sort, asc stand for ascending,desc stand for descending. default asc

## Sample

- To count the number employees in each department ,and group by the feild name dept\_no,and sort in desc order by the field name.

```
db.exec("select dept_no, count(emp_no) as 员工总数 from foo.bar group by dept_no order by dept_no desc")
```



Note:

likesum , count , min , max , avg these counting function must use alias.

## sql limit

limit

Be used to limit the number of returned records.

Grammar

```
limit<limit_num>
```

<limit\_num> : limit number

Sample

- This sample will return the first 10 records from the result set.

```
db.exec("select * from foo.bar limit 10")
```

## sql offset

offset

Be used to set the number of skipped records.

Grammar

```
offset<offset_num>
```

<offset\_num> : the number of skipped records

Sample

- This sample will skip the first five records, then return from the sixth record.

```
db.exec("select * from foo.bar offset 5")
```

## sql as

As

Be used to specify alias for the collections or field names.

Grammar

```
<cs_name.cl_name | (select_set) | field_name> AS <alias_name>
```

<cs\_name> : collectionspace name

<cl\_name> : collection name

select\_set : result set

field\_name : field name

<alias\_name>: alias name

### Sample

- specify alias for collection

```
db.exec("select T1.age,T1.name from foo.bar as T1 where T1.age>10")
```

- specify alias for field

```
db.exec("select age as 年齡 from foo.bar where age>10")
```

- specify alias for result set

```
db.exec("select T.age,T.name from (select age,name from foo.bar) as T where T.age>10")
```

## sql inner join

### inner join

According to two or more collections in the relationship between the field name,inner join is used to query data from those collections.

### Grammar

```
<collection1_name | (select-set1) as <alias1_name>
inner join
<collection2_name | (select-set2)> as <alias2_name>
[ON condition]
```

### Sample

- There have employee information table foo.emp and departmenrt information table foo.dept, query the employee number filed emp\_no in which department name field dept\_name.

```
db.exec("select E.emp_no,D.dept_name from foo.emp as E inner join foo.dept as D on
E.dept_no=D.dept_no")
```



#### Note:

1.can't contain non-union conditions,the following is a wrong way.

```
select T1.a,T2.b from foo.bar1 as T1 inner join foo.bar2 as T2 on T1.a<10
```

2.can't use 'select \*' statement in join layer.

## sql left outer join

### left outer join

left outer join will return all the records from the left collection name(collection1\_name),even though there is no matched records in the right collection name(collection2\_name).

### Grammar

```
<collection1_name | (select-set1) as <alias1_name>
left outer join
<collection2_name | (select-set2)> as <alias2_name>
[ON condition]
```

### Sample

- There have employee information table foo.emp and departmenrt information table foo.dept, query the employee number filed emp\_no in which department name field dept\_name.

```
db.exec("select E.emp_no,D.dept_name from foo.emp as E left outer join foo.dept as D on
E.dept_no=D.dept_no where D.dept_no<4")
```

## sql right outer join

### right outer join

right outer join will return all the records from the right collection name(collection2\_name),even though there is no matched records in the left collection name(collection1\_name).

### Grammar

```
<collection1_name | (select_set1) as <alias1_name>
right outer join
<collection2_name | (select_set2)> as <alias2_name>
[ON condition]
```

### Sample

- There have employee information table foo.emp and departmenrt information table foo.dept, query the employee number filed emp\_no<10 in which department name field dept\_name.

```
db.exec("select E.emp_no,D.dept_name from foo.emp as E right outer join foo.dept as D on
E.dept_no=D.dept_no where E.emp_no<10")
```

## sql sum()

### sum()

Be used to sum for the specified field name.

### Grammar

```
sum(field_name) as <alisa_name>
```



Note: 1.when use sum() function,you must use aliases.

2.automatically skipped for the non-numeric field.

### Sample

- This sample will sum for the field name age in the collection bar.

```
db.exec("select sum(age) as 年龄总和 from foo.bar")
```

## sql count()

### count() function

Be used to count, return the number of records that match the specified field.

### Grammar

```
count(field_name) as <alisa_name>
```



Note: 1.When use count function to count , you must use alisa name .

### Sample

- counting for the field name age in the collection bar.

```
db.exec("select count(age) as amount from foo.bar")
```



## sql avg()

avg() function

Be used to calculate average for all values of the specified field.

Grammar

```
avg(field_name) as <alisa_name>
```



Note: 1.When use avg function to calculate average for the field , you must use alisa name .  
2.It will auto skip for non-numeric .

Sample

- calculating average for the field name age in the collection bar.

```
db.exec("select avg(age) as 平均年龄 from foo.bar")
```

## sql max()

max() function

Be used to return the maximum value of the specified field name.

Grammar

```
max(field_name) as <alisa_name>
```



Note: 1.when use max() to return the max value of the specified field name,you must use aliases.

Sample

- Return the max value of the field nameage.

```
db.exec("select max(age) as 最大年龄 from foo.bar")
```

## sql min()

min() function

Be used to return the min value of the specified field name.

Grammar

```
min(field_name) as <alisa_name>
```



Note: 1.when use min() to return the min value of the specified field name,you must use aliases.

Sample

- Return the min value of the field name age.

```
db.exec("select min(age) as 最小年龄 from foo.bar")
```

## sql first()

first() function

The choice of the first data.

Grammar

```
first(field name)
```

Example

- Select the first data table

Original records in the table

```
{a: 1, b: 2}
```

```
{a: 2, b: 3}
```

```
{a: 3, b: 3}
```

```
SELECT FIRST(a) AS a, b FROM foo.bar GROUP BY b
```

Get record

```
{a: 1, b: 2}
```

```
{a: 2, b: 3}
```

## sql last()

last() function

The choice of the last data.

Grammar

```
last(field name)
```

Example

- Finally, a data selection table

Original records in the table

```
{a: 1, b: 2}
```

```
{a: 2, b: 3}
```

```
{a: 3, b: 3}
```

```
SELECT LAST(a) AS a, b FROM foo.bar GROUP BY b
```

Get record

```
{a: 1, b: 2}
```

```
{a: 3, b: 3}
```

## sql push()

push() function

The fields in multiple records into a single array.

Grammar

```
push(field name)
```

### Example

- The table fields in multiple records into an array

```
Original records in the table
{a: 1, b: 1}
{a: 2, b: 2}
{a: 2, b: 3}

SELECT a, PUSH(b) AS b FROM foo.bar GROUP BY a

Get record
{a: 1, b: [1]}
{a: 2, b: [2, 3]}
```

## sql addtoaset()

### addtoaset() function

The fields in multiple records into an array of no duplicate values.

### Grammar

```
addtoaset(field name)
```

### Example

- The table fields in multiple records into an array of no duplicate values

```
Original records in the table
{a: 1, b: 1}
{a: 2, b: 2}
{a: 2, b: 3}
{a: 2, b: 3}

SELECT a, ADDTOSET(b) AS b FROM foo.bar GROUP BY a

Get record
{a: 1, b: [1]}
{a: 2, b: [2, 3]}
```

## sql buildobj()

### buildobj() function

The record multiple fields into a single object.

### Grammar

```
buildobj(field name1,fieldname2,...)
```

### Example

- The table records in multiple fields into a single object

```
Original records in the table
{a: 1, b: 1, c: 1}
{a: 2, b: 2, c: 2}
{a: 3, b: 3, c: 3}

SELECT a, buildobj(b,c) AS d FROM foo.bar

Get record
```

```
{a: 1, d: {b: 1, c: 1}}
{a: 2, d: {b: 2, c: 2}}
{a: 3, d: {b: 3, c: 3}}
```

sql mergearrayset()

mergearrayset() function

Multiple arrays into one field that does not contain an array of repeating fields.

Grammar

```
mergearrayset (field name)
```

Example

- The table multiple arrays into an array of fields does not contain duplicate fields

```
Original records in the table
{a: 1, b: [1, 2, 3]}
{a: 1, b: [2, 2, 3]}

SELECT a, MERGEARRAYSET(b) AS b FROM foo.bar GROUP BY a

Get record
{a: 1, b: [1, 2, 3]}
```

Mapping Table from SQL to SequoiaDB

Concepts and terms

SQL	SequoiaDB
database	database
table	collection
row	document / BSON document
column	field
index	index
table joins	embedded documents
primary key (Users can take any unique column as primary key.)	primary key (In SequoiaDB, primary key is automatically as the field called "_id".)

Create and Alter

The following table shows create and alter statements on table level in SQL and the corresponding ones in SequoiaDB.

SQL Statements	Sequoiadb Statements	Relative Link
create table student (id not null,stu_id varchar(50), age number primary key(id))	The system will automatically generate a collection when data is firstly inserted. If the value on "_id" is not specified, the system will automatically generate a "_id" value. "db.collectionspace.student({stu_id:"01",age:20})" Of course, you can also create a collection manually "db.collectionspace.createCL("student")"	<a href="#">insert(),createCL()</a>

SQL Statements	Sequoiadb Statements	Relative Link
alter table student add name varchar(50)	In a collection, the structure is not changable, because there is no relative manipulation to describe or change the structure. But the method update() can add new field to document record with "\$set". "db.collectionspace.student.update({},{\$set:{name:"Tom"}})"	<a href="#">update(),\$set</a>
alter table student drop column name	In a collection, the structure is not changable, because there is no relative manipulation to describe or change the structure. But the method update() can delete existing field from document record with "\$unset". "db.collectionspace.student.update({},{\$unset:{name:"Tom"}})"	<a href="#">update(),\$unset</a>
create index index_stu_id on student (stu_id)	db.collectionspace.student.createIndex("index_stu_id",{stu_id:-1})	<a href="#">createIndex(),index</a>
drop table student	db.collectionspace.dropCL("student")	<a href="#">dropCL()</a>

## Insert

The following table shows insert statement on table level in SQL and the corresponding one in SequoiaDB.

SQL Statements	SequoiaDB Statements	Relative Link
insert into student(stu_id,age) values("01",20)	db.collectionspace.student.insert({stu_id:"01",age:20})	<a href="#">insert()</a>

## Select

The following table shows read statement on table level in SQL and the corresponding one in SequoiaDB.

SQL Statements	SequoiaDB Statements	Relative Link
select * from student	db.collectionspace.student.find()	<a href="#">find()</a>
select stu_id,age from student	db.collectionspace.student.find({}, {stu_id:"01",age:20})	<a href="#">find()</a>
select * from student where age > 25	db.collectionspace.student.find({age:{\$gt:25}})	<a href="#">find(),\$gt</a>
select age from student where age = 25 and stu_id = "01"	db.collectionspace.student.find({age:25,stu_id:"01"}, {age:25})	<a href="#">find()</a>
select count(*) from student	db.collectionspace.student.count()	<a href="#">count()</a>
select count(stu_id) from student	db.collectionspace.student.count({stu_id:{\$exists:1}})	<a href="#">count(),\$exists</a>

## Update

The following table shows update statement on table level in SQL and the corresponding one in SequoiaDB.

SQL Statements	Sequoiadb Statements	Relative Link
update student set age = 25 where stu_id = "01"	db.collectionspace.student.update({stu_id:"01"}, {\$set:{age:25}})	<a href="#">update(),\$set</a>
update student set age = age + 2 where stu_id = "01"	db.collectionspace.student.update({stu_id:"01"}, {\$inc:{age:2}})	<a href="#">update(),\$inc</a>

## Delete

The following table shows delete statement on table level in SQL and the corresponding one in SequoiaDB.

SQL Statements	Sequoiadb Statements	Relative Link
delete from student where age = 20	db.collectionspace.student.remove({age:20})	<a href="#">remove()</a>
delete from student	db.collectionspace.student.remove()	<a href="#">remove()</a>

## Limits

### Document

Description	Limits
Minimum length of document	At least contains one field
Maximum length of document	When document is transformed into BSON, the length should be lesser than 16777168 bytes.
Field name	It should not be started with "\$". It should not contain ".".

### Collection

Description	Limits
Maximum length of collection name	127 bytes
Collection name	It should not be started with "\$" or "SYS". It should not contain ".".
Maximum capacity of collection consisted of single node	It is the maximum capacity of collection space
Maximum amount of collections in a collection space consisted of single node	4096

### Collection Space

Description	Limits
Maximum length of collection space name	127 bytes
Collection space name	It should not be started with "\$" or "SYS". It should not contain ".".
Size of data page	4096, 8192, 16384, 32768, 65536
Maximum capacity of collection space consisted of single node	It can be 512GB, 1TB, 2TB, 4TB, 8TB.
Maximum amount of standalone-node collection space	4096

### Index

Description	Limits
Maximum length of each data index key	1024 bytes
Total length of index (It includes index name, index key, etc.) .	When an index is transformed into BSON, the length should be equal to or lesser than the size of data page minus 48 bytes.
Multiple index	It can contain all kinds of legal fields. But it contains at most one field which contains array.
Order value of index key	1 or -1
Maximum amount of index in one collection	64

### Database

Description	Limits
Minimum size of log file	64MB

Description	Limits
Maximum size of log file	2GB

## Node

Description	Limits
Maximum amount of nodes in each replset	7
Create node	Nodes should be created through hostname rather than IP address.
Network	All the systems in a cluster should be able to visit each other through hostname.
Vote condition of master node	Vote will not be carried out unless more than half of nodes in replset take part in it.

## Shard

Description	Limits
Data split	Each moment, only a range of data can be split in each collection.
Shard key	The value of shard key is unchangeable after data is inserted.
_id	"_id" in shard collection is unique in replset, but maybe not globally unique.
Unique index	It should contain all fields in sharding key.

## Driver

Description	Limits
Thread-safe	Each connection object and its subobject are not thread-safe to each other. Different connections are thread-safe to each other.

## Error Code List

Description	Error Code
IO Exception	-1
Out of Memory	-2
Permission Error	-3
File Not Exist	-4
File Exist	-5
Invalid Argument	-6
Invalid size	-7
Interrupt	-8
hit end of file	-9
system error	-10
has no space	-11
EDU status is not valid	-12
Timeout error	-13
Database is quiesced	-14
Network error	-15
Network is closed from remote	-16
Database is in shutdown status	-17
Application is forced	-18
Given path is not valid	-19
Unexpected file type specified	-20
There's no space for DMS	-21
collection already exist	-22
collection does not exist	-23
user record is too big	-24

Description	Error Code
record does not exist	-25
remote overflow record exist	-26
invalid record	-27
storage unit need reorg	-28
end of collection	-29
context is already opened	-30
context is closed	-31
option is not supported yet	-32
collection space already exist	-33
collection space not exist	-34
storage unit file is invalid	-35
context not exist	-36
more than one field has array	-37
duplicate key exist	-38
key is too large	-39
index extent has no space	-40
index key not exist	-41
hit max number of index	-42
failed to initialize index	-43
collection is dropped	-44
two records get same key and rid	-45
duplicate index name	-46
index name doesn't exist	-47
index flag is unexpected	-48
hit end of index	-49
hit max of dedup buffer	-50
invalid predicates	-51
index is no longer exist	-52
index hint is not valid	-53
no more temp tables available	-54
exceed max number of SU	-55
\$id index can't be dropped	-56
log was not found in log buf	-57
log was not found in log file	-58
replication group not exist	-59
replication group exist	-60
invalid request id is received	-61
session ID does not exist	-62
system edu can't be forced	-63
database is not connected	-64
unexpected result received	-65
corrupted record	-66
backup has already started	-67
backup not completed	-68
backup in progress mode	-69
backup file is damaged	-70
there's no primary found	-71
the requested node not exist	-72
engine help argument is specified	-73
connection state is not valid	-74
invalid handle	-75
client object is freed	-76
transfer has already listened	-77
can not listen specified addr	-78
cannot connect to specified addr	-79
connection does not exist	-80
failed to send	-81
timer id not found	-82



Description	Error Code
route info not found	-83
broken msg	-84
invalid net handle	-85
reorg file is not valid	-86
reorg file is in read only mode	-87
collection status is not valid	-88
collection is not in reorg state	-89
replication group is not activated	-90
the member is not in this group	-91
collection flag not compatible	-92
stroage unit version not compatible	-93
local group version is expired	-94
page size is not valid	-95
remote group version is expired	-96
failed to create a poll	-97
log record is corrupted	-98
given lsn greater than latest	-99
received unknown message	-100
updated information is same as old one	-101
unknow message	-102
empty heap	-103
not primary	-104
data node not enough	-105
data node have no catalog info	-106
data node catlog version old	-107
coord node catalog version old	-108
exceeds the max group size	-109
failed to sync log	-110
failed to replay log	-111
error http struct	-112
failed to negotiate	-113
failed to change dps metadata	-114
SME is corrupted	-115
application is interrupted	-116
application is disconnected	-117
character encoding errors	-118
get failed msg from the node	-119
buffer array is full	-120
sub context is conflict	-121
coord received EOC message	-122
DPS file size are not the same	-123
dps file not recognise	-124
no resource	-125
invalid lsn	-126
data to pipe is too big	-127
catalog auth failed	-128
node is full sync	-129
catnode failed assign data-node	-130
php driver internal error	-131
failed to send the message	-132
there is no node-group register in catalogue-node	-133
remote-node disconnected	-134
couldn't find the match catalogue-info	-135
update catalog failed	-136
received invalid request which opcode is unknowned	-137
coord couldn't find the group-info in local	-138
dms extent is corrupted	-139
remote cluster manage failed	-140

Description	Error Code
remote engines have been stopped partially	-141
sdb service is starting	-142
sdb service has already been started	-143
sdb service is restarting	-144
sdb node is already existed	-145
sdb node is not existed	-146
unable to lock	-147
dms state is not compatible with current command	-148
rebuild has already started	-149
rebuild in progress mode	-150
there is no data in the coord-node's cache	-151
errors occur during the process of evalution	-152
group already exist	-153
group does not exist	-154
node does not exist	-155
failed to start the node	-156
node'configure conflicts	-157
group is empty	-158
The operation is for coord node only	-159
failed to operate on node only	-160
The mutex job already exist	-161
The specified job does not exist	-162
The catalog information is corrupted	-163
\$shard index can't be dropped	-164
The command can't be run in the node	-165
The command can't be run in the serice plane	-166
The group info not exist	-167
Group name is conflict	-168
The collection is not sharded	-169
The record does not contains valid sharding key	-170
A task that already exists does not compatible with the new task	-171
The collection does not exists on the specified group	-172
The specified task does not exist	-173
The record contains more than one sharding key	-174
The mutex task already exist	-175
The split key is not valid or not in the source group	-176
The unique index must include all fields in sharding key	-177
Sharding key cannot be updated	-178
authority is forbidden	-179
There is no catalog address specified by user	-180
current record has been deleted	-181
search condition cannot match any records	-182
index page is reorged and the pos got different lchild	-183
There are duplicate field name exists in the record	-184
Too many records to be inserted at once	-185
Sort-Merge Join only supports equal predicates	-186
Trace is already started	-187
Trace buffer does not exist	-188
Trace file is not valid	-189
transaction-lock is incompatible	-190
system is doing rollback	-191
Bad record when doing sdb import	-192
repeat var name was found	-193
column field is ambiguous	-194
have an error in sql syntax	-195
invalid transactional operation	-196
append to lock-wait-queue	-197
record is deleting	-198

Description	Error Code
index is dropped or invalid	-199
repeat to create catalog-group	-200
parse json error	-201
parse CSV error	-202
log file is out of size	-203
can not remove the only node in the group	-204
need to manually complete the cleanup	-205
can not remove node or group of catalog	-206
group is not exist	-207
can not remove the group with data in it	-208
end of queue	-209
collection has not sharding index, can not split by percent	-210
the param field not exist	-211
too many trace break points are specified	-212
prefetchers are all busy	-213
domain not exist	-214
domain already exist	-215
group is not in domain	-216
sharding type is not hash	-217
split percent is lower	-218
task already finished, can not be canceled	-219
collection is loading	-220
load is error, and rollback	-221
routeID is different from the local	-222
service already exists	-223
not find field	-224
csv field line end	-225
unknown file type	-226
not all nodes are successful to export configuration	-227
this node is not primary and has no data	-228
secret value not same with data unit	-229
engine version argument is specified	-230
sdb help argument is specified	-231
sdb version argument is specified	-232
store procedure is not exist	-233
illegal remove sub-collection	-234
duplicate link sub-collection	-235
invalid main-collection	-236
new boundary is conflict with the existing boundary	-237
new boundary is invalid	-238
hit the high water mark	-239
backup already exist	-240
backup not exist	-241
invalid sub-collection	-242
task has canceled	-243
the sharding-type of main-collection must be range	-244
there is no valid sharding-key field	-245
the operation not support on main-collection	-246
redefine index	-247
Deleting the CS	-248
Reached the maximum number of nodes	-249
the node is business-failures	-250
the node info is expired	-251
wait secondary nodes sync the operation failed	-252
transaction is disabled	-253
datasource had run out	-254
too many open file description	-255
Domain is not empty	-256

Description	Error Code
Rest recv size greater than max size	-257
Wrong data for building bson	-258
out of bound	-259
rest common unknow	-260
successfully done on catalog, but sth wrong happened when did it on data group.	-261
domain does not have any groups at all	-262
suggest sdb om's user to change the passwd	-263