MongoDB 索引

MongoDB 索引

索引通常能够极大的提高查询的效率，如果没有索引，MongoDB在读取数据时必须扫描集合中的每个文件并选取那些符合查询条件的记录。

这种扫描全集合的查询效率是非常低的，特别在处理大量的数据时，查询可以要花费几十秒甚至几分钟，这对网站的性能是非常致命的。

索引是特殊的数据结构，索引存储在一个易于遍历读取的数据集合中，索引是对数据库表中一列或多列的值进行排序的一种结构

MongoDB使用 **ensureIndex()** 或者**createIndex()** 方法来创建索引。

>db.COLLECTION_NAME.ensureIndex({KEY:1})

语法中 Key 值为你要创建的索引字段，1为指定按升序创建索引，如果你想按降序来创建索引指定为-1即可。

>db.col.ensureIndex({"title":1})

ensureIndex() 接收可选参数，可选参数列表如下：

Parameter	Type	Description
background	Boolean	建索引过程会阻塞其它数据库操作，background可指定以后台方式创建索引，即增加 "background" 可选参数。 "background" 默认值为false。
unique	Boolean	建立的索引是否唯一。指定为true创建唯一索引。默认值为false.
name	string	索引的名称。如果未指定，MongoDB的通过连接索引的字段名和排序顺序生成一个索引名称。
dropDups	Boolean	在建立唯一索引时是否删除重复记录,指定 true 创建唯一索引。默认值为 false.
partialFilterExpressionv	Document	设定索引只对满足条件的文档起作用
sparse	Boolean	对文档中不存在的字段数据不启用索引；这个参数需要特别注意，如果设置为true的话，在索引字段中不会查询出不包含对应字段的文档.。默认值为 false.
expireAfterSeconds	integer	指定一个以秒为单位的数值，完成 TTL设定，设定集合的生存时间。
v	indexversion	索引的版本号。默认的索引版本取决于mongod创建索引时运行的版本。
weights	document	索引权重值，数值在 1 到 99,999 之间，表示该索引相对于其他索引字段的得分权重。
default_language	string	对于文本索引，该参数决定了停用词及词干和词器的规则的列表。 默认为英语
language_override	string	对于文本索引，该参数指定了包含在文档中的字段名，语言覆盖默认的language，默认值为 language.

在后台创建索引：

db.col.ensureIndex({open: 1, close: 1}, {background: true})

创建索引：

db.col.ensureIndex({"username":1})

db.col.ensureIndex({"username":1, "age":-1})

该索引被创建后，基于username和age的查询将会用到该索引

或者是基于username的查询也会用到该索引

但是只是基于**age**的查询将不会用到该复合索引

因此，如果想用到复合索引，必须在查询条件中包含复合索引中的前**N**个索引列

然而如果查询条件中的键值顺序和复合索引中的创建顺序不一致的话，**MongoDB**可以智能的帮助我们调整该顺序，以便使复合索引可以为查询所用。

```
db.user.dropIndexes()
db.user.createIndex({"age":-1,"username":1,"sex":1})

db.user.find({"age":"1"}).explain("executionStats") //第1条查询
db.user.find({"sex":"1"}).explain("executionStats") //第2条查询
db.user.find({"age":"1","sex":1}).explain("executionStats")//第3条查询
db.user.find({"username":"1","sex":1}).explain("executionStats")//第4条查询
```

第1条以及第3条查询返回的都是winningPlan.stage = **FETCH**

而第2条和第4条查询返回的都是winningPlan.stage = **COLLSCAN**

查看索引：

db.col.getIndexes()

删除索引：

```
> db.col.dropIndex({"username":1})
> db.col.dropIndexes()
```

排序中使用索引字段

单键索引与查询时排序字段的设置对查询性能的影响

排序字段不包含索引字段，会在内存中排序 **winningPlan.stage = SORT**

排序字段同时包含索引字段和非索引字段 **winningPlan.stage = SORT**

排序字段只包含一个单个索引字段 **winningPlan.stage = FETCH**

排序字段包含多个单个索引字段 **winningPlan.stage = SORT**

在多个字段上做排序时需要使用复合索引

复合索引与排序字段设置对查询性能的影响

```
db.user.find().sort({"age":-1}).explain("executionStats");
db.user.find().sort({"age":1}).explain("executionStats");
db.user.find().sort({"username":1}).explain("executionStats");
db.user.find().sort({"sex":1}).explain("executionStats");
db.user.find().sort({"age":-1,"username":1}).explain("executionStats");
db.user.find().sort({"username":1,"sex":1}).explain("executionStats");
db.user.find().sort({"age":-1,"sex":1}).explain("executionStats");
```

第1,2,5 返回的winningPlan.stage= FETCH ； 其他都是=sort

这说明排序字段只包含创建复合索引字段中的部分字段时排序键的顺序必须和它们在索引中的排列顺序 一致，且不能跳跃(即第一个字段必须有，且不能跳过中间的字段)

```
db.user.find().sort({"age":1,"username":-1}).explain("executionStats")
db.user.find().sort({"age":-1,"username":-1}).explain("executionStats")
```

第1条返回的winningPlan.stage= FETCH;而第2条winningPlan.stage= SORT;这说明sort中指定的所有键的排序顺序(例如递增/递减）必须和索引中的对应键的排序顺序 完全相同，或者 完全相反

可以指定在索引的所有键或者部分键上排序。但是，排序键的顺序必须和它们在索引中的排列顺序 一致，sort中指定的所有键的排序顺序(例如递增/递减）必须和索引中的对应键的排序顺序 完全相同,或者 完全相反

稀疏索引与$exists的关系

在一个字段上创建稀疏索引，索引会跳过所有不包含被索引键(字段)的文档
在执行查询｛被索引键:{$exists:false}},不会使用该稀疏索引，除非显示指定hint({被索引键:1})

```
db.scores.insert({"userid":"lqj"})
db.scores.insert({"userid":"fj","score":90})
db.scores.insert({"userid":"sys","score":80})
//在字段score上创建稀疏索引
db.scores.createIndex( { score: 1 } , {sparse:true} , {name:"IDX_SCORE"} )
db.scores.find( { score:{$exists:false}}).explain("executionStats")
db.scores.find( { score:{$exists:false}}).hint({score:1}).explain("executionStats")
db.scores.find( { score:{$exists:true}}).explain("executionStats")
```

第1条在执行$exists:false 查询时，其返回的winningPlan.stage=COLLSCAN(表示扫描全表)；
第2条 exists:false 查询时，显示指定了hint，其返回的winningPlan.stage=FETCH(使用索引扫描)；
第3条 exists:true 查询，没有显示指定hint，其返回的winningPlan.stage=FETCH(使用索引扫描)；

在某一个被创建了稀疏索引字段上执行exists:false查询时，需要显示指定hint，其索引才会起作用；而执行 exists:true查询时，则不需要。
在字段上创建普通索引 ，如果文档不含该字段这其索引值会被设为null，而稀疏索引会跳过该文档；这就是说使用该索引扫描集合时稀疏索引会比普通索引少。

局部索引(Partial Indexes)

1. 这是3.2.0版本以后新增的
2. 只会对Collections满足条件partialFilterExpression的文档进行索引
3. 使用该索引，会在一定程度上减少存储空间和创建索引和维护性能降低的成本

语法： db.collection.createIndex({“字段名”: 1或-1,…”字段名n”: 1或-1},{“partialFilterExpression”:{partialFilterExpression }})
partialFilterExpression表达式如下
equality expressions (i.e. field: value or using the $eq operator)
$exists: true expression,
$gt, $gte, $lt, $lte expressions,
$type expressions,
$and operator at the top-level only
示例：

```
db.restaurants.createIndex(
   { cuisine: 1, name: 1 },
   { partialFilterExpression: { rating: { $gt: 5 } } }
)
```

TTL索引

1. TTL 集合支持失效时间设置，当超过指定时间后，集合自动清除超时的文档，这用来保存一些诸如session会话信

息的时候非常有用，或者存储缓存数据使用。但删除会有延时

2. 索引的字段必须是一个日期的 bson 类型

3. 你不能创建 TTL 索引，如果要索引的字段已经在其他索引中使用。否则超时后文档不会被自动清除。

4. 索引不能包含多个字段