MongoDB

## MongoDB 概念解析

|  |  |  |
| --- | --- | --- |
| **SQL术语/概念** | **MongoDB术语/概念** | **解释/说明** |
| database | database | 数据库 |
| table | collection | 数据库表/集合 |
| row | document | 数据记录行/文档 |
| column | field | 数据字段/域 |
| index | index | 索引 |
| table joins |  | 表连接,MongoDB不支持 |
| primary key | primary key | 主键,MongoDB自动将\_id字段设置为主键 |



## 数据库

一个mongodb中可以建立多个数据库。

MongoDB的默认数据库为"db"，该数据库存储在data目录中

"show dbs" 命令可以显示所有数据的列表。



"db" 命令可以显示当前数据库对象或集合。

"use"命令，可以连接到一个指定的数据库。即切换数据库.

数据库名命名规则：

* 不能是空字符串（"")。
* 不得含有' '（空格)、.、$、/、\和\0 (空字符)。
* 应全部小写。
* 最多64字节。

有一些数据库名是保留的，可以直接访问这些有特殊作用的数据库。

* **admin**： 从权限的角度来看，这是"root"数据库。要是将一个用户添加到这个数据库，这个用户自动继承所有数据库的权限。一些特定的服务器端命令也只能从这个数据库运行，比如列出所有的数据库或者关闭服务器。
* **local:** 这个数据永远不会被复制，可以用来存储限于本地单台服务器的任意集合
* **config**: 当Mongo用于分片设置时，config数据库在内部使用，用于保存分片的相关信息。

## 数据类型

|  |  |
| --- | --- |
| **数据类型** | **描述** |
| String | 字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。 |
| Integer | 整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。 |
| Boolean | 布尔值。用于存储布尔值（真/假）。 |
| Double | 双精度浮点值。用于存储浮点值。 |
| Min/Max keys | 将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。 |
| Arrays | 用于将数组或列表或多个值存储为一个键。 |
| Timestamp | 时间戳。记录文档修改或添加的具体时间。 |
| Object | 用于内嵌文档。 |
| Null | 用于创建空值。 |
| Symbol | 符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。 |
| Date | 日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。 |
| Object ID | 对象 ID。用于创建文档的 ID。 |
| Binary Data | 二进制数据。用于存储二进制数据。 |
| Code | 代码类型。用于在文档中存储 JavaScript 代码。 |
| Regular expression | 正则表达式类型。用于存储正则表达式。 |

## 创建数据库

use 数据库名

若数据库不存在，则创建数据库

## 删除文档

db.collection.remove(

<query>,

{

justOne: <boolean>,

writeConcern: <document>

}

)

参数说明：

query :（可选）删除的文档的条件。

justOne : （可选）如果设为 true 或 1，则只删除一个文档。默认是全删

writeConcern :（可选）抛出异常的级别。

删除所有（类似truncate）：

db.col.remove({})

## 查询文档

数组查询：

* 1. 数组普通查询

假如type是["mongodb", "javascript"]：

// mongodb在数组type中即可

db.article.find({"type": "mongodb"})

//type数组中只要有一个满足大于4，即可匹配

db.article.find({"type": { $gt : 4}})

//数组type中，至少有一个元素满足后面的条件,同上面类似

也可以用在返回内容的筛选上；

db.article.find({"type": { "$elemMatch": {$eq : “mongodb”}}})

db.article.find({"kown": { "$elemMatch": {a: 1, b: {"$gt": 2}}}})

* 1. 多个元素的数组查询

db.article.find({"type": {"$all": ["mongodb", "javascript"]}})

必须同时包含"mongodb", "javascript"的数组type，才能匹配

区别与：db.article.find({"type":["mongodb", "javascript"]})，它代表完全匹配，而且顺序不能变。

* 1. 限制数组长度查询

db.article.find({"type": {"$size": 2}})

* 1. 是否存在

//字段love是否存在

db.article.find({"love": {"$exists": true}})

* 1. 正则表达式

db.article.find({"title": /mongodb/i}) // i是忽略大小写

* 1. 内嵌文档（点语法）

//{address: { name: "nanji" }}

db.article.find({"address.name": "nanji"})

数组也可以使用点语法：

//{comments: [{title: "mongodb"}, {title: "javascript"}]}

db.article.find({"comments.title": "mongodb"})

## $type数据类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类型** | | **数字** | | **备注** |
| Double | 1 | |  | |
| String | 2 | |  | |
| Object | 3 | |  | |
| Array | 4 | |  | |
| Binary data | 5 | |  | |
| Undefined | 6 | | 已废弃。 | |
| Object id | 7 | |  | |
| Boolean | 8 | |  | |
| Date | 9 | |  | |
| Null | 10 | |  | |
| Regular Expression | 11 | |  | |
| JavaScript | 13 | |  | |
| Symbol | 14 | |  | |
| JavaScript (with scope) | 15 | |  | |
| 32-bit integer | 16 | |  | |
| Timestamp | 17 | |  | |
| 64-bit integer | 18 | |  | |
| Min key | 255 | | Query with -1. | |
| Max key | 127 | |  | |

db.col.find({"title" : {$type : 2}})

## 分析查询

explain()

db.users.find({gender:"M"},{user\_name:1,\_id:0}).explain(‘executionStats’)

* indexOnly的true值表示此查询已使用索引。
* cursor字段指定使用的游标的类型。BTreeCursor类型表示使用了索引，并且还给出了使用的索引的名称。 BasicCursor表示完全扫描，而不使用任何索引的情况。
* n表示返回的文档数。
* nscannedObjects表示扫描的文档总数。
* nscanned表示扫描的文档或索引条目的总数。
* totalKeysExamined索引扫描条数
* totalDocsExamined文档扫描条数

理想情况，nReturned=totalKeysExamined & totalDocsExamined=0

不通过文档查询，全部使用索引

sort的顺序跟创建索引的顺序必须一致，否则不会生效！！！

索引是{filed1: 1, field2:1}， sort({fileld1: 1, filed2: 1}) 可命中索引，sort({filed2:1,filed1:1})索引无效

hint()

$hint操作符强制查询优化器使用指定的索引来运行查询。当要测试具有不同索引的查询的性能时，这就特别有用了。 例如，以下查询指定要用于此查询的gender和user\_name字段的索引

> db.users.find({gender:"M"},{user\_name:1,\_id:0}).hint({gender:1,user\_name:1})

要使用$explain来分析上述查询 -

>db.users.find({gender:"M"},{user\_name:1,\_id:0}).hint({gender:1,user\_name:1}).explain()

## 原子操作

MongoDB不支持多文档原子事务。 但是，它可以为单个文档提供了原子操作。

如果文档有一百个字段，则更新语句将要么全部更新，要么全部不更新所选字段的值，因此在原始级别保持原子性。

维持原子性的推荐方法是将所有相关信息保存在一起，并使用嵌入式文档在一个文档中一起更新。

|  |
| --- |
| {  "\_id":1,  "product\_name": "Huawei P9",  "category": "mobiles",  "product\_total": 5,  "product\_available": 3,  "product\_bought\_by": [  {  "customer": "Kobe",  "date": "2017-07-08"  },  {  "customer": "Maxsu",  "date": "2018-07-28"  }  ]  } |

当有新顾客购买时，先检查product\_available是否大于0，若大于0则减少product\_available,并将新顾客的信息插入到product\_bought\_by中。

|  |
| --- |
| >db.products.findAndModify({  query:{\_id:2,product\_available:{$gt:0}},  update:{  $inc:{product\_available:-1},  $push:{product\_bought\_by:{customer:"Curry",date:"2017-08-08"}}  }  }) |

若不适用嵌入式，而将库存和购物者信息放在不同集合中，可能会造成0库存但还能购买的情况。因为这种操作不是原子性的。

## ObjectId 解析

\_id: ObjectId(' 4e7020cb7cac81af7136236b ')

4e7020cb7cac81af7136236b 这个24位字符串，是一组十六进制字符串，每个字节2位的十六进制，所以一共是12个字节；



一共有四部分：

1. time

前4个字节 " 4e7020cb ",转成十进制，即：" 1315971275 "，是个时间戳

2）machine

接下来是" 7cac81 ",代表所在主机的唯一标识符，一般是机器主机名的散列值，确保了不同主机生成不同的机器hash值，确保在分布式中不造成冲突，所以在同一个机器上生成的ObjectId中间的字符串都是一样的

3）pid

进程Id，确保同一台机器不同的mongodb进程产生的ObjectId不冲突，接下来的两位" af71 "，就是pid

4）inc

自增计数器，前面的九个字节是保证了一秒内不同机器不同进程生成objectId不冲突，这后面的三个字节“36236b”是一个自动增加的计数器，用来确保在同一秒内产生的objectId也不会发现冲突，允许256的3次方等于16777216条记录的唯一性。