MongoDB教育訓練-07

- 只能保證分片唯一，不能保證全局唯一
- sparse index：欄位可能不存在document中使用
- partial index：依據欄位的值決定要不要放入索引（刪除的不放）

Index Options

- Indexes can enforce a unique constraint.
  `db.a.createIndex({custid: 1}, { unique: true})`
  - NULL is a value and so only one record can have a NULL in unique field.
- Sparse Indexes don't index missing fields or nulls.
  `db.scores.createIndex( { score: 1 } , { sparse: true } )`
  - Sparse Indexes are superseded by Partial Indexes
  - Use `{ field : { $exists : true } }` for your partialFilterExpression.
- Partial indexes index a subset of documents based on values.
  - Can greatly reduce index size

```
db.orders.createIndex( { customer: 1, store: 1 },
    { partialFilterExpression: { archived: false } } )
```

- hashed indexes：md5 取前20個byte，並不可作為唯一索引，使用時機：控制索引大小（如url）
- 關於時間不適合，會造成分散在所有分片

Hashed Indexes

- Hashed Indexes index a 20 byte md5 of the BSON value.
  - They support exact match only.
  - They cannot be used for unique constraints.
  - They can potentially reduce index size if original values are large.
  - Downside: random values in a BTree use excessive resources.

```
db.people.createIndex({ name : "hashed"})
```

- 索引增加讀的效能，減低寫的效能
- 建議每張表的索引< 10
- 新增資料會做索引異動，更新不會(但更新欄位會)

Indexes and Performance

- Indexes improve read performance when used.
- Each index adds ~10% overhead
  - Hashed Indexes can add a lot more.
- An index is modified any time a document:
  - Is inserted (applies to all indexes)
  - Is deleted (applies to all indexes)
  - Is updated in such a way that its indexed field changes

索引越大，記憶體越大

Use Indexes with Care

- Every query should use an index.
- Every index should be used by a query.
- Indexes require RAM.
- Be mindful about the choice of key.

前綴壓縮

- App le
- App le iphon
- App le store

Index Prefix Compression

- MongoDB Indexes use a special compressed format
  - Each entry is just delta from the previous one
  - If there are identical entries, they need only one byte
- As indexes are inherently sorted, this makes them much smaller
- Smaller indexes mean less RAM required to keep them in RAM

Multikey indexs

建在array上的index (是不是由系統判斷)

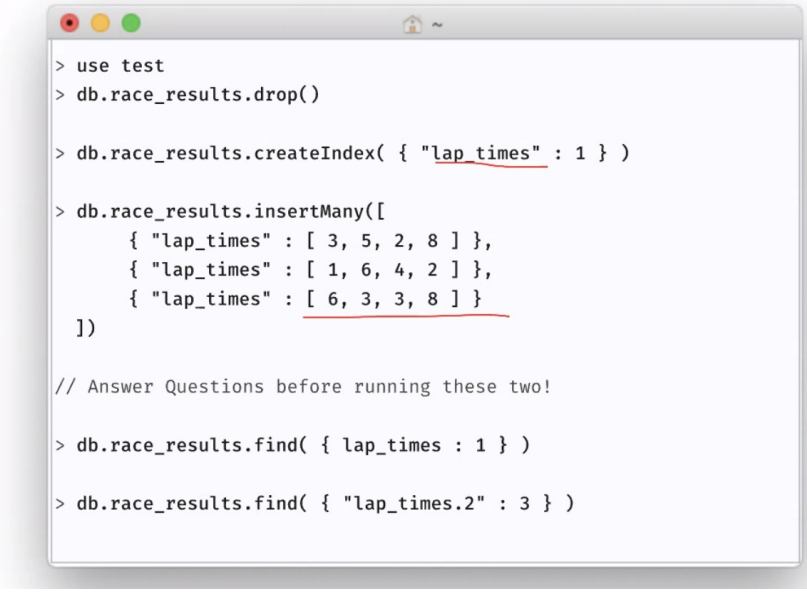
Introduction to Multikey Indexes

- A multikey index is an index that has indexed an array.
- An index entry is created on each **unique** value found in an array.
- Multikey indexes can index primitives, documents, or sub-arrays.
- There is nothing special that you need to do to create a multikey index.
- You create them using createIndex() just as you would with an ordinary single-field index.
- If any field in the index is ever found to be an array then the index is described as being multikey.

Multikey Basics

Exercise for the class

- How many records are we inserting?
- How many index entries are there in total for lap_times?
- How many records will the queries find?
- Will they use our index?



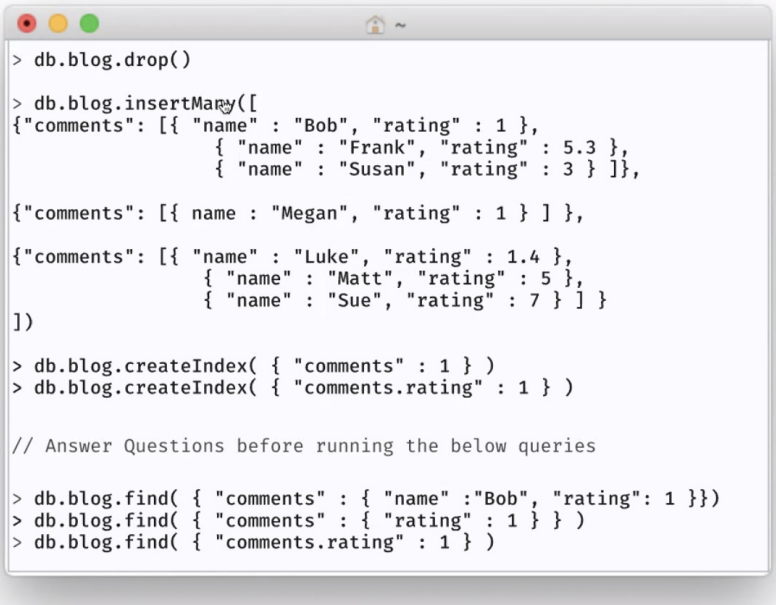
通常使用"comments.rating"：1 這種索引

Array of Documents

How Many Documents in total?

For each query:

- How many results?
- Which index, if any, will it use?



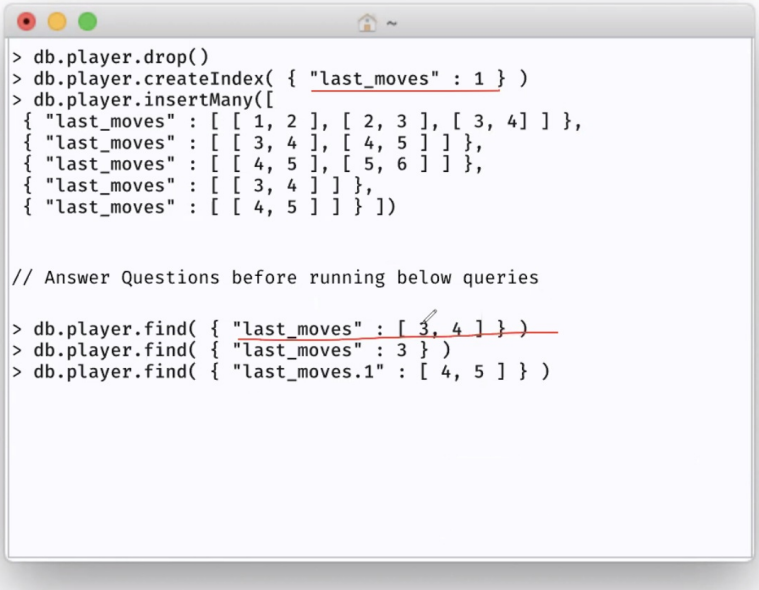
```
> db.blog.drop()

> db.blog.insertMany([
{"comments": [{ "name" : "Bob", "rating" : 1 },
                { "name" : "Frank", "rating" : 5.3 },
                { "name" : "Susan", "rating" : 3 } ]},

{"comments": [{ name : "Megan", "rating" : 1 } ] },

{"comments": [{ "name" : "Luke", "rating" : 1.4 },
                { "name" : "Matt", "rating" : 5 },
                { "name" : "Sue", "rating" : 7 } ] }
])

> db.blog.createIndex( { "comments" : 1 } )
> db.blog.createIndex( { "comments.rating" : 1 } )


// Answer Questions before running the below queries

> db.blog.find( { "comments" : { "name" :"Bob", "rating": 1 }})
> db.blog.find( { "comments" : { "rating" : 1 } } )
> db.blog.find( { "comments.rating" : 1 } )
```

Arrays of Arrays

How Many Documents in total?

For each query:

- How many results?
- Which index, if any, will it use?



```
> db.player.drop()
> db.player.createIndex( { "last_moves" : 1 } )
> db.player.insertMany([
 { "last_moves" : [ [ 1, 2 ], [ 2, 3 ], [ 3, 4] ] },
 { "last_moves" : [ [ 3, 4 ], [ 4, 5 ] ] },
 { "last_moves" : [ [ 4, 5 ], [ 5, 6 ] ] },
 { "last_moves" : [ [ 3, 4 ] ] },
 { "last_moves" : [ [ 4, 5 ] ] } ])


// Answer Questions before running below queries

> db.player.find( { "last_moves" : [ 3, 4 ] } )
> db.player.find( { "last_moves" : 3 } )
> db.player.find( { "last_moves.1" : [ 4, 5 ] } )
```

Compound符合索引

多欄位索引 (欄位A & 欄位B 查詢適用)

Compound Indexes

- Create an index based on more than one field.
  - They are called Compound Indexes
  - MongoDB normally only uses one index per query
  - Compound indexes are the most common type of indexes
  - They are the same conceptually as used in an RDBMS
- You may use up to 32 fields in a compound index.
- The field order and direction is **very** important.
- You create them like a single field index but with more fields specified

```
db.people.createIndex({lastname:1, firstname:1, score:1})
```

Compound Indexes 複合索引

- MongoDB 通常使用一個索引作查詢
- 最多可以做32個
- 順序很重要

Compound Indexes

- Create an index based on more than one field.
  - They are called Compound Indexes
  - MongoDB normally only uses one index per query
  - Compound indexes are the most common type of indexes
  - They are the same conceptually as used in an RDBMS
- You may use up to 32 fields in a compound index.
- The field order and direction is **very** important.
- You create them like a single field index but with more fields specified

```
db.people.createIndex({lastname:1, firstname:1, score:1})
```

Compound Indexes

- An Index can be used as long as the **first field in index is in the query**.
- Other fields in the Index do not **need** to be in the query.

```
createIndex({country:1,state:1,city:1})
find({country:"UK",city:"Glasgow"})
```

Uses an Index for country and city but must look at every state in the country, so looks at many index keys.

```
createIndex({country:1,city:1,state:1})
```

A Better Index for this query as can go straight to country and city.
The directions matter when doing range queries.

欄位順序必須相同

The Order of Fields Matters

- Equality First.
  - In order of selectivity
  - What fields, for a typical query, will filter the most.
  - selectivity != cardinality, selective can be a boolean choice
  - Normally Male/Female is not selective (for the common query case)
  - Dispatched versus Delivered IS selective though
- Then Range or Sort (Usually Sort)
  - Sorts are much more expensive than range queries when no index is used.

Example: A Simple Message Board

We will look at the indexes needed for a simple Message Board App.

We want to automatically clean up our board and remove some older, low-rated anonymous messages on a regular basis.

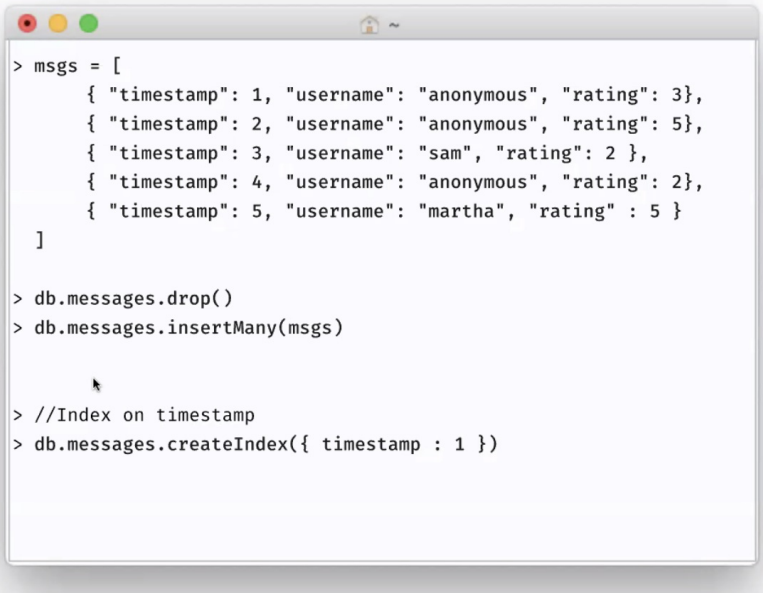
Here is our query requirement:

1. Find all messages in a specified timestamp range.
2. Select for whether the messages are anonymous or not.
3. Sort by rating from lowest to highest.

Message Board Index

Here we create five messages, just the relevant fields.

And an index on timestamp.

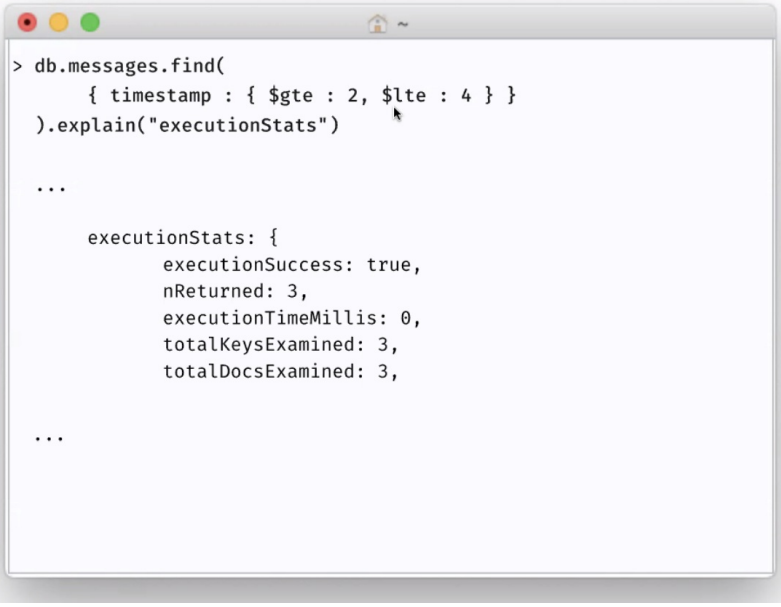


```
> msgs = [
      { "timestamp": 1, "username": "anonymous", "rating": 3},
      { "timestamp": 2, "username": "anonymous", "rating": 5},
      { "timestamp": 3, "username": "sam", "rating": 2 },
      { "timestamp": 4, "username": "anonymous", "rating": 2},
      { "timestamp": 5, "username": "martha", "rating" : 5 }
  ]

> db.messages.drop()
> db.messages.insertMany(msgs)


> //Index on timestamp
> db.messages.createIndex({ timestamp : 1 })
```

Message Board Index

Explain shows good performance, but this is not our whole query.

Need to filter for anonymous users



```
> db.messages.find(
      { timestamp : { $gte : 2, $lte : 4 } }
  ).explain("executionStats")

  ...

      executionStats: {
            executionSuccess: true,
            nReturned: 3,
            executionTimeMillis: 0,
            totalKeysExamined: 3,
            totalDocsExamined: 3,

  ...
```

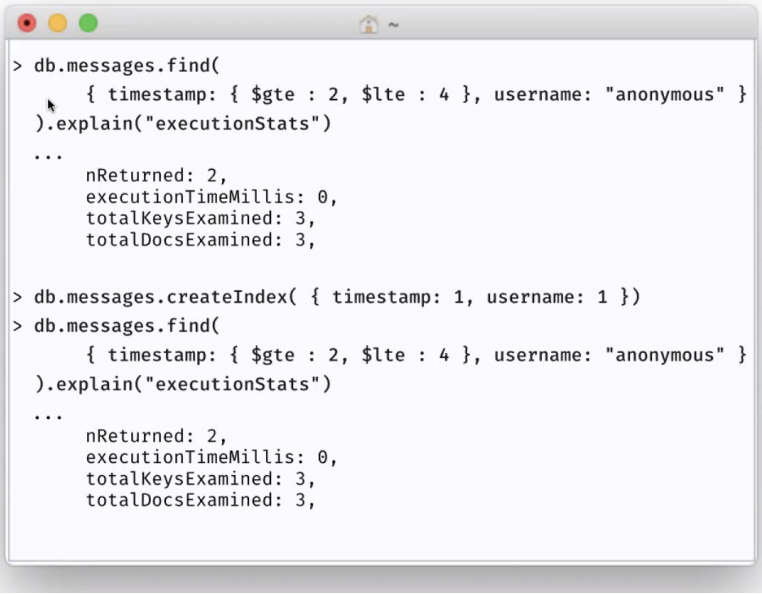
Message Board Index

Not as efficient as it could be:

totalKeysExamined > nReturned

What if we add username to the index?

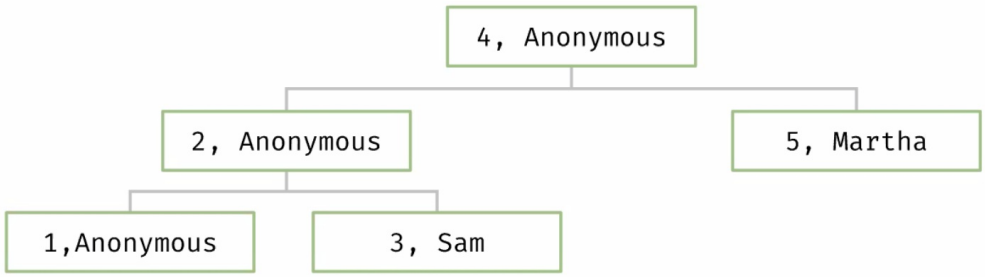
No improvement?



```
> db.messages.find(
      { timestamp: { $gte : 2, $lte : 4 }, username: "anonymous" }
  ).explain("executionStats")
  ...
      nReturned: 2,
      executionTimeMillis: 0,
      totalKeysExamined: 3,
      totalDocsExamined: 3,

> db.messages.createIndex( { timestamp: 1, username: 1 })
> db.messages.find(
      { timestamp: { $gte : 2, $lte : 4 }, username: "anonymous" }
  ).explain("executionStats")
  ...
      nReturned: 2,
      executionTimeMillis: 0,
      totalKeysExamined: 3,
      totalDocsExamined: 3,
```

Index in wrong order

```
Query: {timestamp:{$gte:2, $lte:4}, username:"anonymous"}
Index: { timestamp: 1, username: 1 }
```



1. Must do range part first as the timestamp is first in the index
2. Start at the first timestamp and check >= 2
3. Walk tree Left to Right until timestamp, not <= 4 (3 nodes) check each if 'anonymous.'
4. Return only 2 of the three nodes visited (2 and 4)

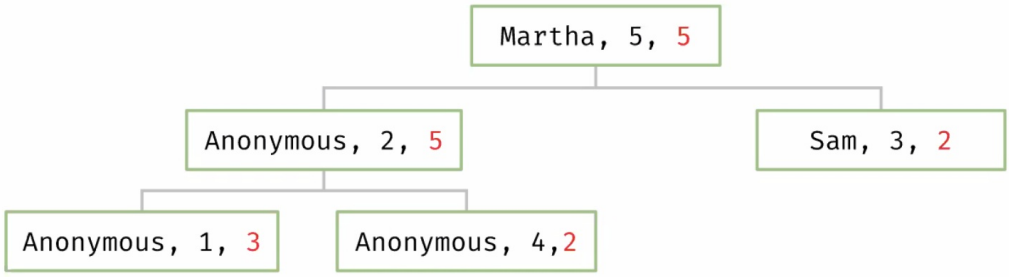
What about the sort?

- Sort by rating from lowest to highest.
  - Order in the index must match sort order
  - Otherwise, we need to reorder
  - Here we have an explicit **SORT** stage

```
> db.messages.find(
      { timestamp: {$gte:2, $lte:4}, username:"anonymous"}
  ).sort({rating:1}).explain("executionStats")

  ...

      executionStats: {
            executionSuccess: true,
            nReturned: 2,
            executionTimeMillis: 0,
            totalKeysExamined: 3,
            totalDocsExamined: 3,
            executionStages: {
                  stage: 'SORT',


  ...
```

Index in correct order ?

```
Query: {timestamp:{$gte:2, $lte:4}, username:"anonymous"}     Sort: { rating: 1 }
With Index: { username: 1, timestamp: 1 , rating:1 }
```



1. Exact Match at start filters down the tree to walk (Just Anonymous).
2. Find first Anonymous where timestamp >= 2
3. Walk tree whilst Anonymous & timestamp <= 4
4. Visits only two index nodes in total (2 and 4)
5. But results in order 5,2 - so need sorted

ＥＳＲ原則

等值 排序 範圍

10000以內不算數據量大

Rules of Compound Indexing

- Equality before range
- Equality before sorting
- Sorting before range

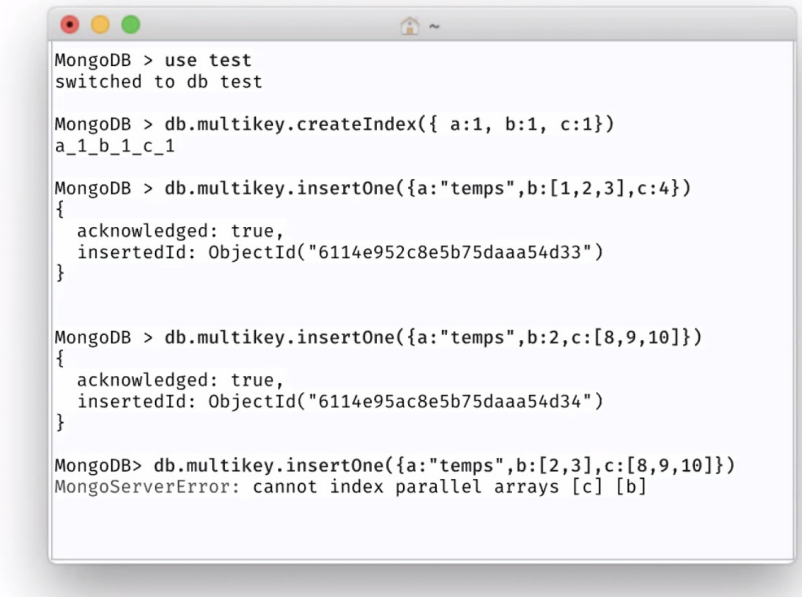
兩欄位都是陣列，禁止使用，因為所有數量是陣列的乘積

MultiKey Compound Indexes

An Index can be Compound (multiple fields) and MultiKey (Multiple values for a field)

In a Document any non _id field can be an array

Indexing two arrays in one document is an error as we would need to store all possible combinations.

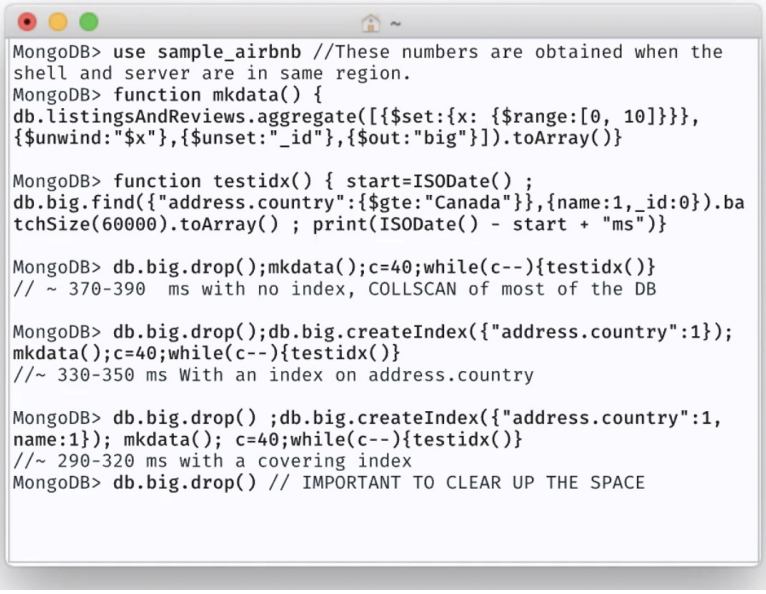


index 完全覆蓋查詢(資料都從index返回，不用直接查詢document)

_id 不要出現, multikey 不適用

Index Covered Queries

- Fetch all data from Index not the Document
- Can be much faster in some cases
- Does have some limitations to be aware of
  - { _id : 0 }
  - no multikey indexes



```
MongoDB> use sample_airbnb //These numbers are obtained when the
shell and server are in same region.
MongoDB> function mkdata() {
db.listingsAndReviews.aggregate([{$set:{x: {$range:[0, 10]}}},
{$unwind:"$x"},{$unset:"_id"},{$out:"big"}]).toArray()}

MongoDB> function testidx() { start=ISODate() ;
db.big.find({"address.country":{$gte:"Canada"}},{name:1,_id:0}).ba
tchSize(60000).toArray() ; print(ISODate() - start + "ms")}

MongoDB> db.big.drop();mkdata();c=40;while(c--){testidx()}
// ~ 370-390  ms with no index, COLLSCAN of most of the DB

MongoDB> db.big.drop();db.big.createIndex({"address.country":1});
mkdata();c=40;while(c--){testidx()}
//~ 330-350 ms With an index on address.country

MongoDB> db.big.drop() ;db.big.createIndex({"address.country":1,
name:1}); mkdata(); c=40;while(c--){testidx()}
//~ 290-320 ms with a covering index
MongoDB> db.big.drop() // IMPORTANT TO CLEAR UP THE SPACE
```

Exercise - Compound indexes

Create the best index you can for this query - how efficient can you get it?

```
> query = { amenities: "Waterfront",
  "bed_type" : { $in : [ "Futon", "Real Bed" ] },
  first_review : { $lt: ISODate("2018-12-31") },
  last_review : { $gt : ISODate("2019-02-28") }
}

> project = { bedrooms:1 , price: 1, _id:0, "address.country":1}
> order = {bedrooms:-1,price:1}
> use sample_airbnb
> db.listingsAndReviews.find(query,project).sort(order)
```

```
db.listingsAndReviews.find({amenities:"Waterfront","bed_type":{$in:["Futon","Real Bed"]}})
.projection({bedrooms:1, price:1,_id:0,"address.country":1}).explain("executionStats")


db.listingsAndReviews.createIndex({amenities:1, bedrooms:-1,price:11,bed_type:1,first_review:1,last_review:1})
```

修訂版本 #4
★由 treeman 建立於 17 ﷽Q﷽G﷽ 2021 18:50:57
✎由 treeman 更新於 5 ﷽﷽﷽ 2023 10:14:59