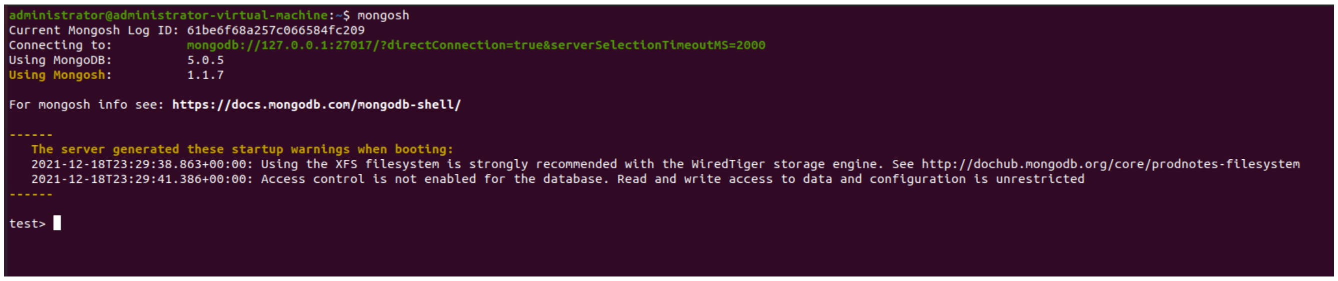
Run MongoDB as a Docker Container:

```
docker run -d -p 27017:27017 --name MONGO_CONTAINER mongo:latest
```

Start the MongoDB shell:

```
docker exec -it MONGO_CONTAINER bash
```

```
mongosh command
```

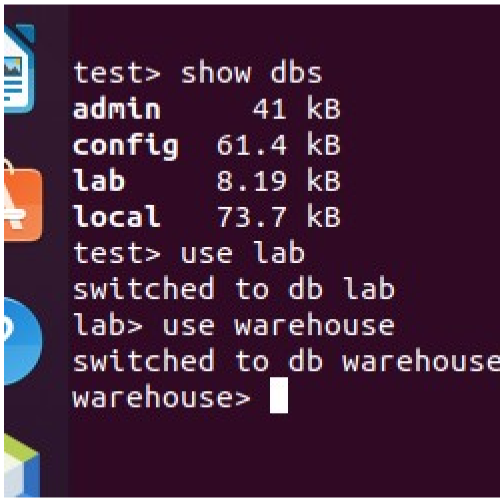


```
administrator@administrator-virtual-machine:~$ mongosh
Current Mongosh Log ID: 61be6f68a257c066584fc209
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:          5.0.5
Using Mongosh:          1.1.7

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting:
   2021-12-18T23:29:38.863+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2021-12-18T23:29:41.386+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test> 
```

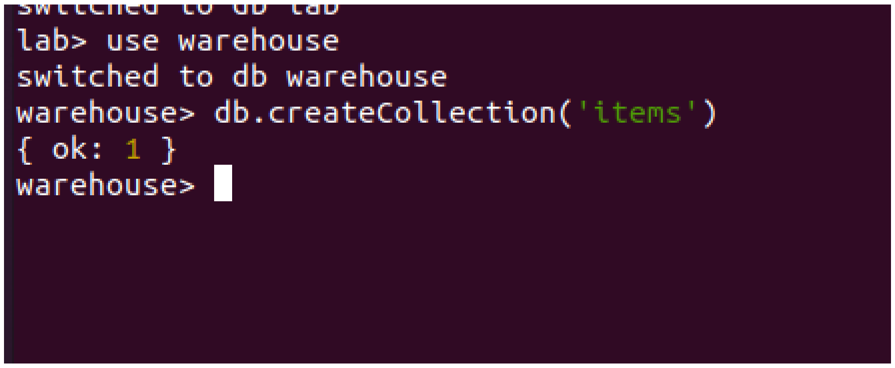


```
test> show dbs
admin     41 kB
config  61.4 kB
lab     8.19 kB
local   73.7 kB
test> use lab
switched to db lab
lab> use warehouse
switched to db warehouse
warehouse> 
```

Show dbs → we select the database.

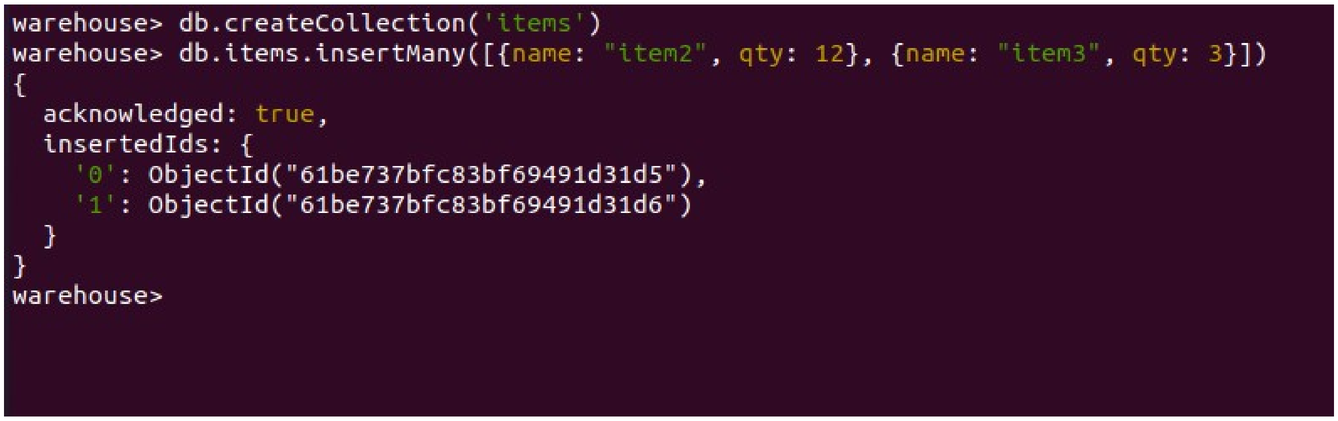
Create and use a new database warehouse.

Create a new collection named Items.



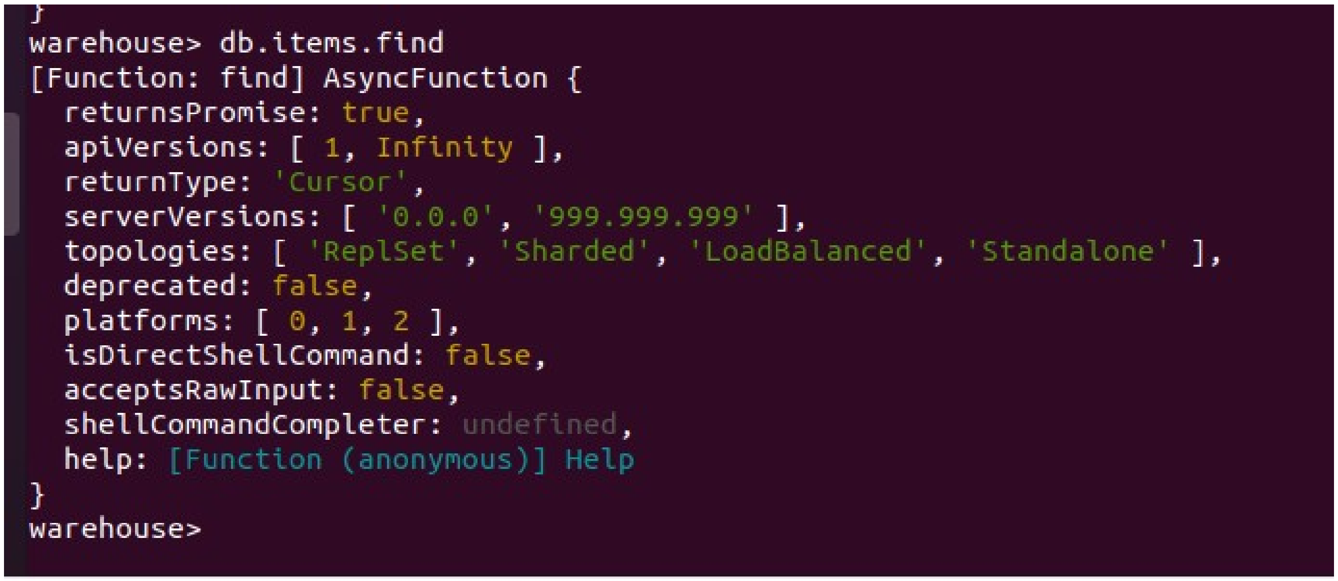
```
switched to db lab
lab> use warehouse
switched to db warehouse
warehouse> db.createCollection('items')
{ ok: 1 }
warehouse> 
```

Add a document in the colection:



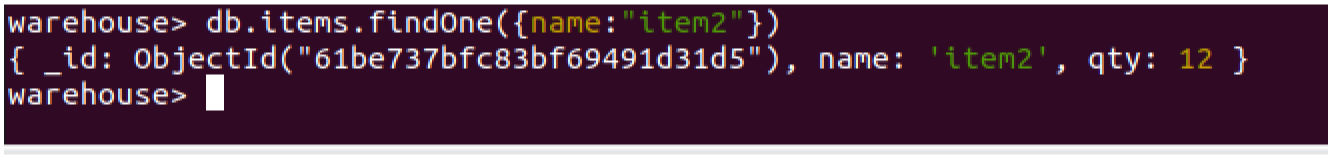
```
warehouse> db.createCollection('items')
warehouse> db.items.insertMany([{name: "item2", qty: 12}, {name: "item3", qty: 3}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("61be737bfc83bf69491d31d5"),
    '1': ObjectId("61be737bfc83bf69491d31d6")
  }
}
warehouse>
```

Show all documents in the collection:



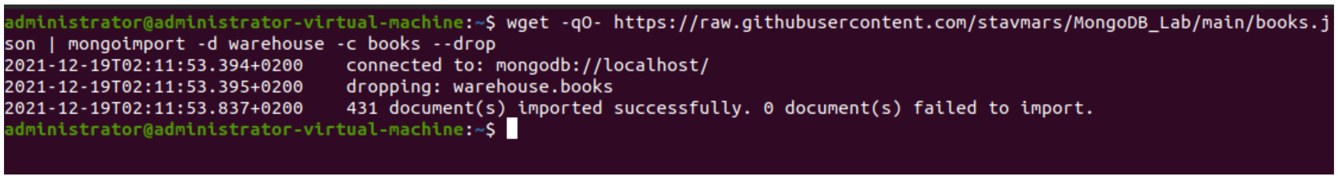
```
}
warehouse> db.items.find
[Function: find] AsyncFunction {
  returnsPromise: true,
  apiVersions: [ 1, Infinity ],
  returnType: 'Cursor',
  serverVersions: [ '0.0.0', '999.999.999' ],
  topologies: [ 'ReplSet', 'Sharded', 'LoadBalanced', 'Standalone' ],
  deprecated: false,
  platforms: [ 0, 1, 2 ],
  isDirectShellCommand: false,
  acceptsRawInput: false,
  shellCommandCompleter: undefined,
  help: [Function (anonymous)] Help
}
warehouse>
```

Find a single document by name:



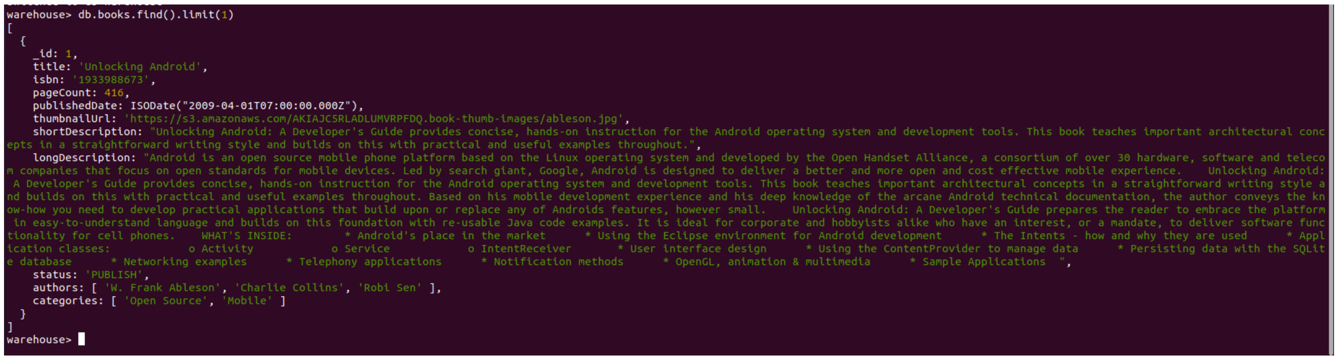
```
warehouse> db.items.findOne({name:"item2"})
{ _id: ObjectId("61be737bfc83bf69491d31d5"), name: 'item2', qty: 12 }
warehouse>
```

Import the books data set.



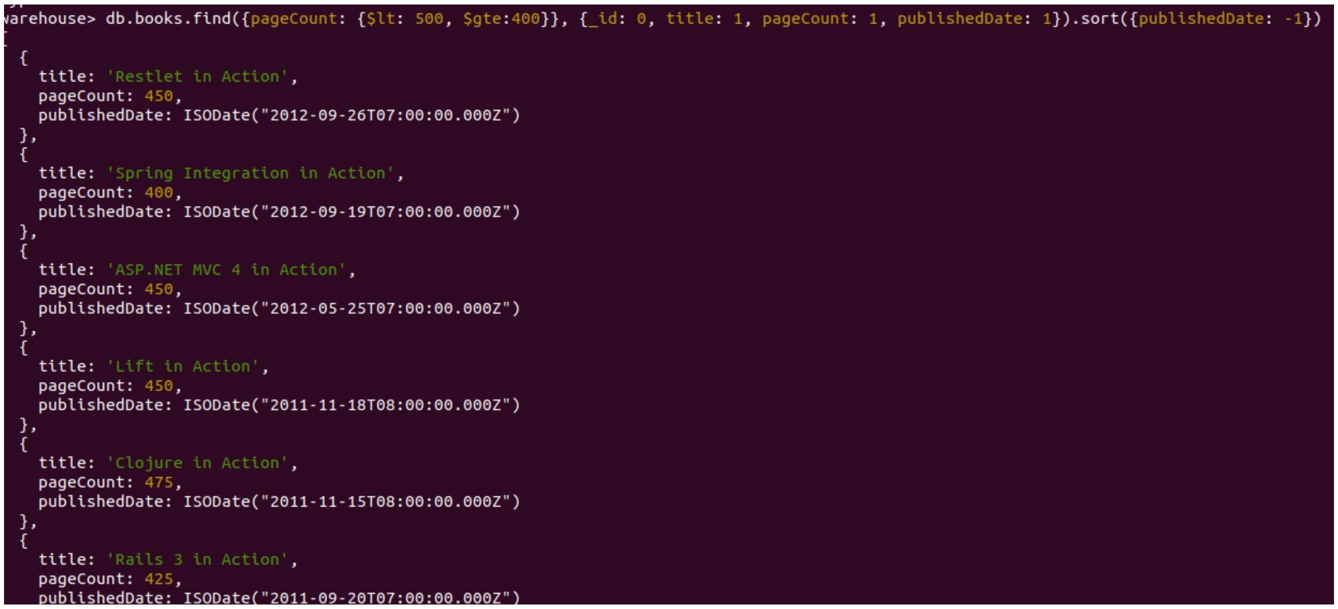
```
administrator@administrator-virtual-machine:~$ wget -qO- https://raw.githubusercontent.com/stavmars/MongoDB_Lab/main/books.j
son | mongoimport -d warehouse -c books --drop
2021-12-19T02:11:53.394+0200    connected to: mongodb://localhost/
2021-12-19T02:11:53.395+0200    dropping: warehouse.books
2021-12-19T02:11:53.837+0200    431 document(s) imported successfully. 0 document(s) failed to import.
administrator@administrator-virtual-machine:~$
```

`db.books.find().limit(1)`

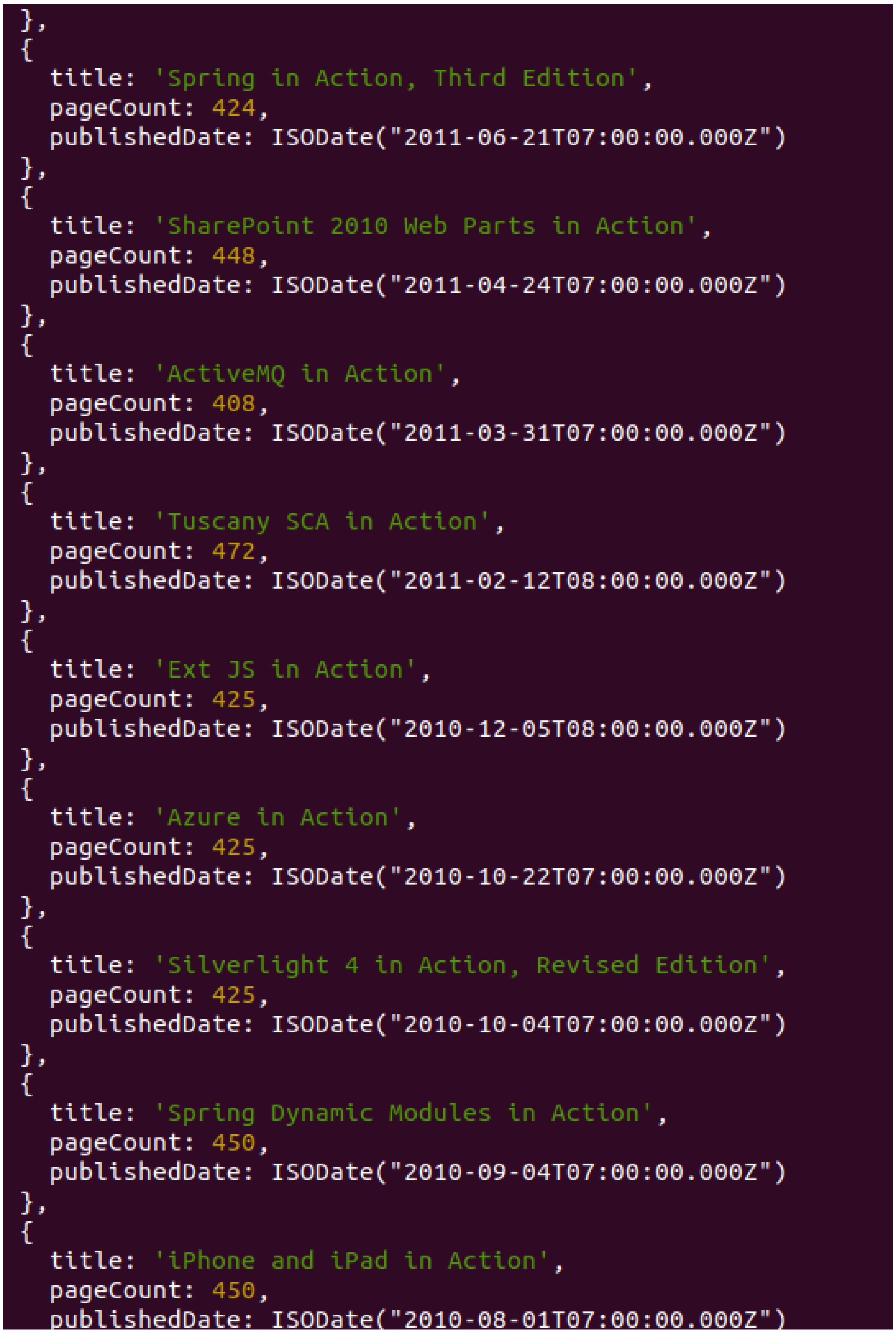


```
warehouse> db.books.find().limit(1)
[
  {
    _id: 1,
    title: 'Unlocking Android',
    isbn: '1933988673',
    pageCount: 416,
    publishedDate: ISODate("2009-04-01T07:00:00.000Z"),
    thumbnailUrl: 'https://s3.amazonaws.com/AKIAJC5RLADLUMVRPFDQ.book-thumb-images/ableson.jpg',
    shortDescription: "Unlocking Android: A Developer's Guide provides concise, hands-on instruction for the Android operating system and development tools. This book teaches important architectural conc
epts in a straightforward writing style and builds on this with practical and useful examples throughout.",
    longDescription: "Android is an open source mobile phone platform based on the Linux operating system and developed by the Open Handset Alliance, a consortium of over 30 hardware, software and teleco
m companies that focus on open standards for mobile devices. Led by search giant, Google, Android is designed to deliver a better and more open and cost effective mobile experience.    Unlocking Android:
 A Developer's Guide provides concise, hands-on instruction for the Android operating system and development tools. This book teaches important architectural concepts in a straightforward writing style a
nd builds on this with practical and useful examples throughout. Based on his mobile development experience and his deep knowledge of the arcane Android technical documentation, the author conveys the kn
ow-how you need to develop practical applications that build upon or replace any of Androids features, however small.    Unlocking Android: A Developer's Guide prepares the reader to embrace the platform
 in easy-to-understand language and builds on this foundation with re-usable Java code examples. It is ideal for corporate and hobbyists alike who have an interest, or a mandate, to deliver software func
tionality for cell phones.    WHAT'S INSIDE:        * Android's place in the market      * Using the Eclipse environment for Android development      * The Intents - how and why they are used      * Appl
ication classes:            o Activity           o Service            o IntentReceiver       * User interface design      * Using the ContentProvider to manage data      * Persisting data with the SQLit
e database      * Networking examples      * Telephony applications      * Notification methods      * OpenGL, animation & multimedia      * Sample Applications  ",
    status: 'PUBLISH',
    authors: [ 'W. Frank Ableson', 'Charlie Collins', 'Robi Sen' ],
    categories: [ 'Open Source', 'Mobile' ]
  }
]
warehouse>
```

To find the books with a number of pages that is greater or equal than 400 but less than 500, sort them by their publication date descending and print their titles, page
counts and date they were published

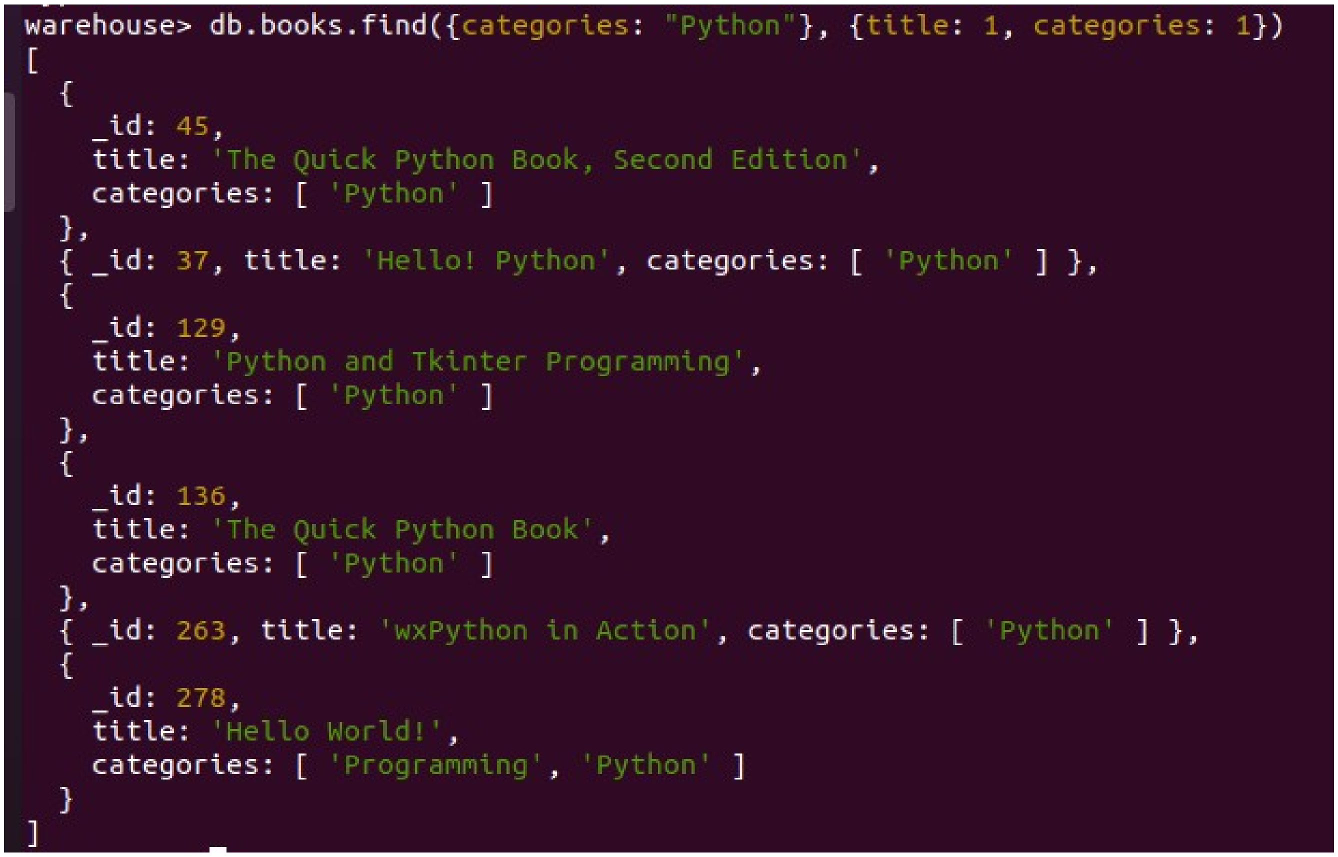


```
warehouse> db.books.find({pageCount: {$lt: 500, $gte:400}}, {_id: 0, title: 1, pageCount: 1, publishedDate: 1}).sort({publishedDate: -1})
[
  {
    title: 'Restlet in Action',
    pageCount: 450,
    publishedDate: ISODate("2012-09-26T07:00:00.000Z")
  },
  {
    title: 'Spring Integration in Action',
    pageCount: 400,
    publishedDate: ISODate("2012-09-19T07:00:00.000Z")
  },
  {
    title: 'ASP.NET MVC 4 in Action',
    pageCount: 450,
    publishedDate: ISODate("2012-05-25T07:00:00.000Z")
  },
  {
    title: 'Lift in Action',
    pageCount: 450,
    publishedDate: ISODate("2011-11-18T08:00:00.000Z")
  },
  {
    title: 'Clojure in Action',
    pageCount: 475,
    publishedDate: ISODate("2011-11-15T08:00:00.000Z")
  },
  {
    title: 'Rails 3 in Action',
    pageCount: 425,
    publishedDate: ISODate("2011-09-20T07:00:00.000Z")
```



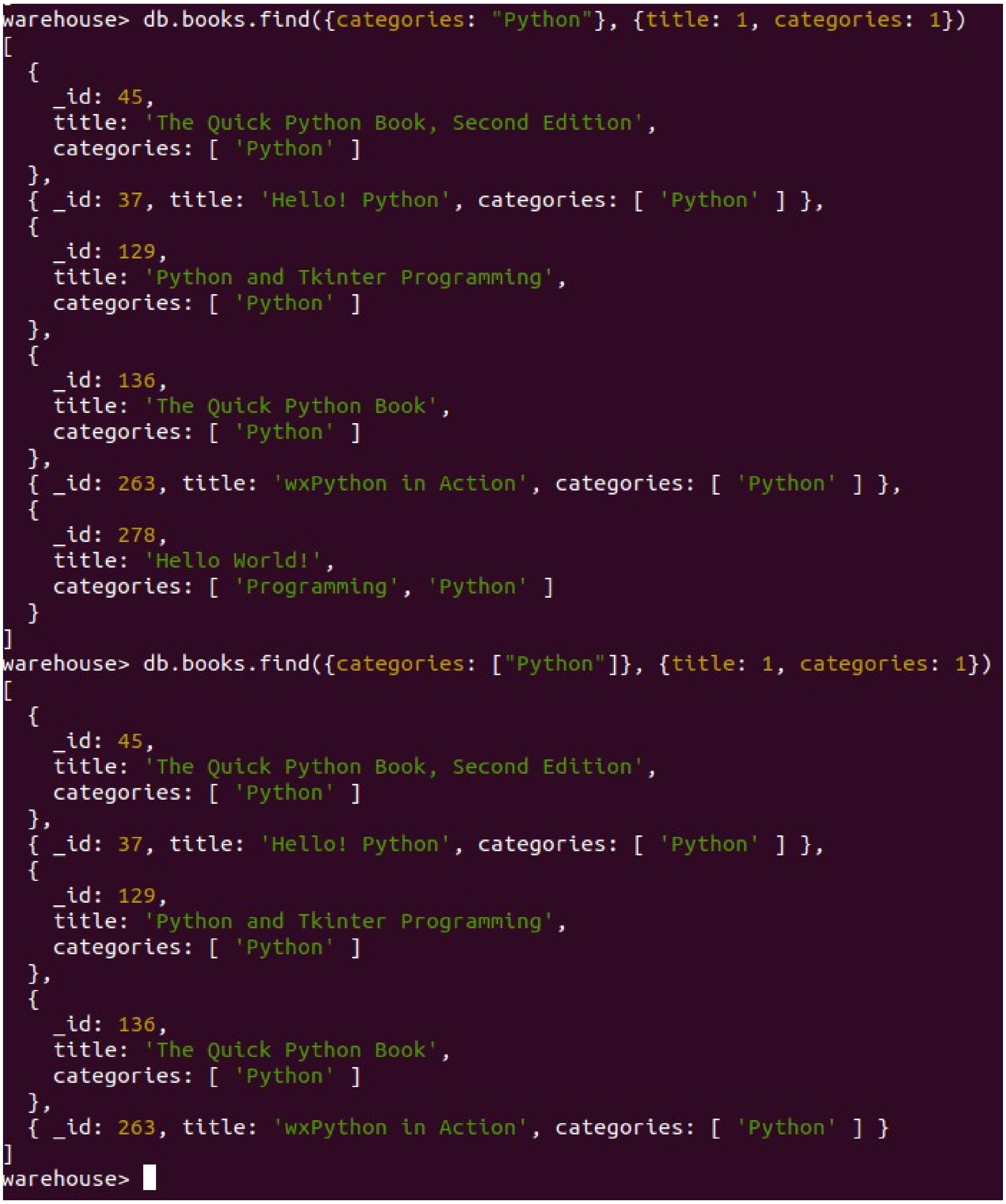
```
},
{
  title: 'Spring in Action, Third Edition',
  pageCount: 424,
  publishedDate: ISODate("2011-06-21T07:00:00.000Z")
},
{
  title: 'SharePoint 2010 Web Parts in Action',
  pageCount: 448,
  publishedDate: ISODate("2011-04-24T07:00:00.000Z")
},
{
  title: 'ActiveMQ in Action',
  pageCount: 408,
  publishedDate: ISODate("2011-03-31T07:00:00.000Z")
},
{
  title: 'Tuscany SCA in Action',
  pageCount: 472,
  publishedDate: ISODate("2011-02-12T08:00:00.000Z")
},
{
  title: 'Ext JS in Action',
  pageCount: 425,
  publishedDate: ISODate("2010-12-05T08:00:00.000Z")
},
{
  title: 'Azure in Action',
  pageCount: 425,
  publishedDate: ISODate("2010-10-22T07:00:00.000Z")
},
{
  title: 'Silverlight 4 in Action, Revised Edition',
  pageCount: 425,
  publishedDate: ISODate("2010-10-04T07:00:00.000Z")
},
{
  title: 'Spring Dynamic Modules in Action',
  pageCount: 450,
  publishedDate: ISODate("2010-09-04T07:00:00.000Z")
},
{
  title: 'iPhone and iPad in Action',
  pageCount: 450,
  publishedDate: ISODate("2010-08-01T07:00:00.000Z")
```

To find all Python books and print their titles and categories we run the following query



```
warehouse> db.books.find({categories: "Python"}, {title: 1, categories: 1})
[
  {
    _id: 45,
    title: 'The Quick Python Book, Second Edition',
    categories: [ 'Python' ]
  },
  { _id: 37, title: 'Hello! Python', categories: [ 'Python' ] },
  {
    _id: 129,
    title: 'Python and Tkinter Programming',
    categories: [ 'Python' ]
  },
  {
    _id: 136,
    title: 'The Quick Python Book',
    categories: [ 'Python' ]
  },
  { _id: 263, title: 'wxPython in Action', categories: [ 'Python' ] },
  {
    _id: 278,
    title: 'Hello World!',
    categories: [ 'Programming', 'Python' ]
  }
]
```

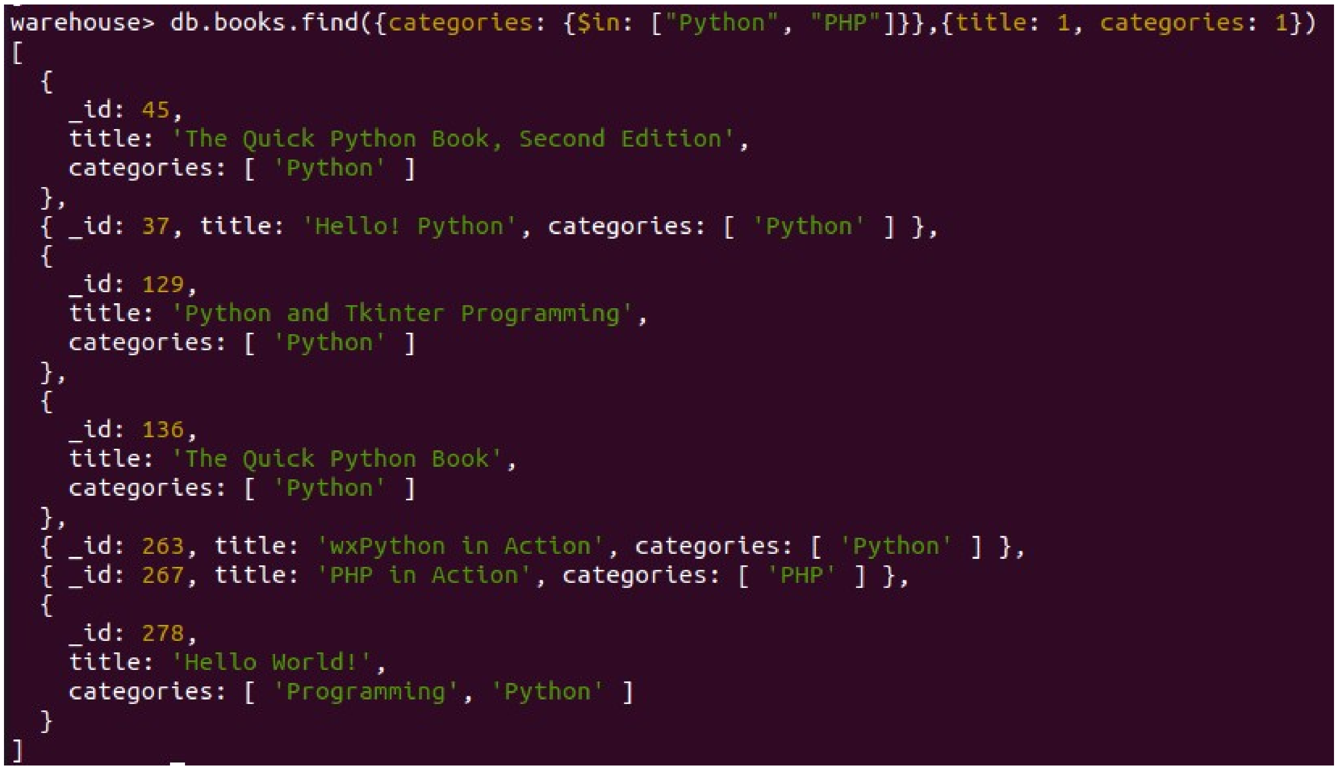
Compare the results of the query above with the following one:



```
warehouse> db.books.find({categories: "Python"}, {title: 1, categories: 1})
[
  {
    _id: 45,
    title: 'The Quick Python Book, Second Edition',
    categories: [ 'Python' ]
  },
  { _id: 37, title: 'Hello! Python', categories: [ 'Python' ] },
  {
    _id: 129,
    title: 'Python and Tkinter Programming',
    categories: [ 'Python' ]
  },
  {
    _id: 136,
    title: 'The Quick Python Book',
    categories: [ 'Python' ]
  },
  { _id: 263, title: 'wxPython in Action', categories: [ 'Python' ] },
  {
    _id: 278,
    title: 'Hello World!',
    categories: [ 'Programming', 'Python' ]
  }
]
warehouse> db.books.find({categories: ["Python"]}, {title: 1, categories: 1})
[
  {
    _id: 45,
    title: 'The Quick Python Book, Second Edition',
    categories: [ 'Python' ]
  },
  { _id: 37, title: 'Hello! Python', categories: [ 'Python' ] },
  {
    _id: 129,
    title: 'Python and Tkinter Programming',
    categories: [ 'Python' ]
  },
  {
    _id: 136,
    title: 'The Quick Python Book',
    categories: [ 'Python' ]
  },
  { _id: 263, title: 'wxPython in Action', categories: [ 'Python' ] }
]
warehouse> 
```

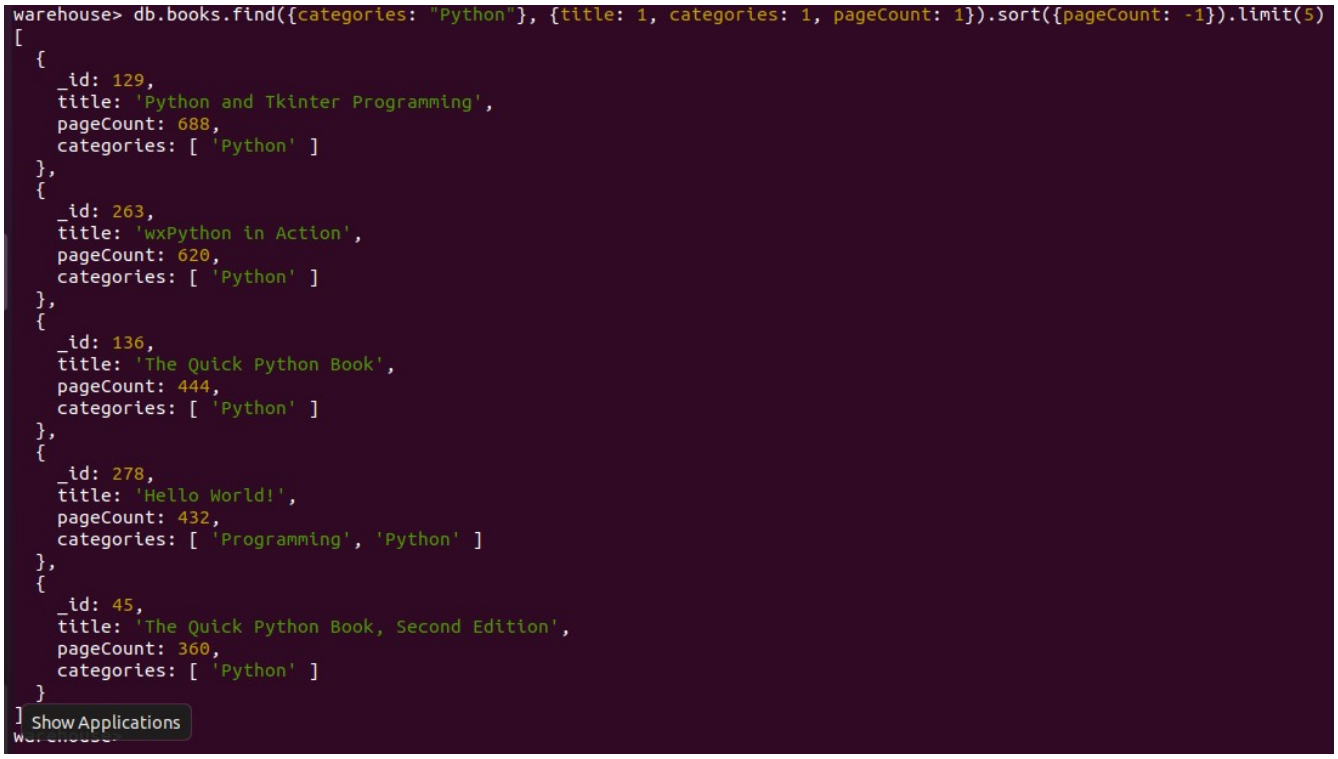
The difference is that in the latter it doesn’t check for the “python” word inside a list.

To find all books that are either about Python or PHP run:



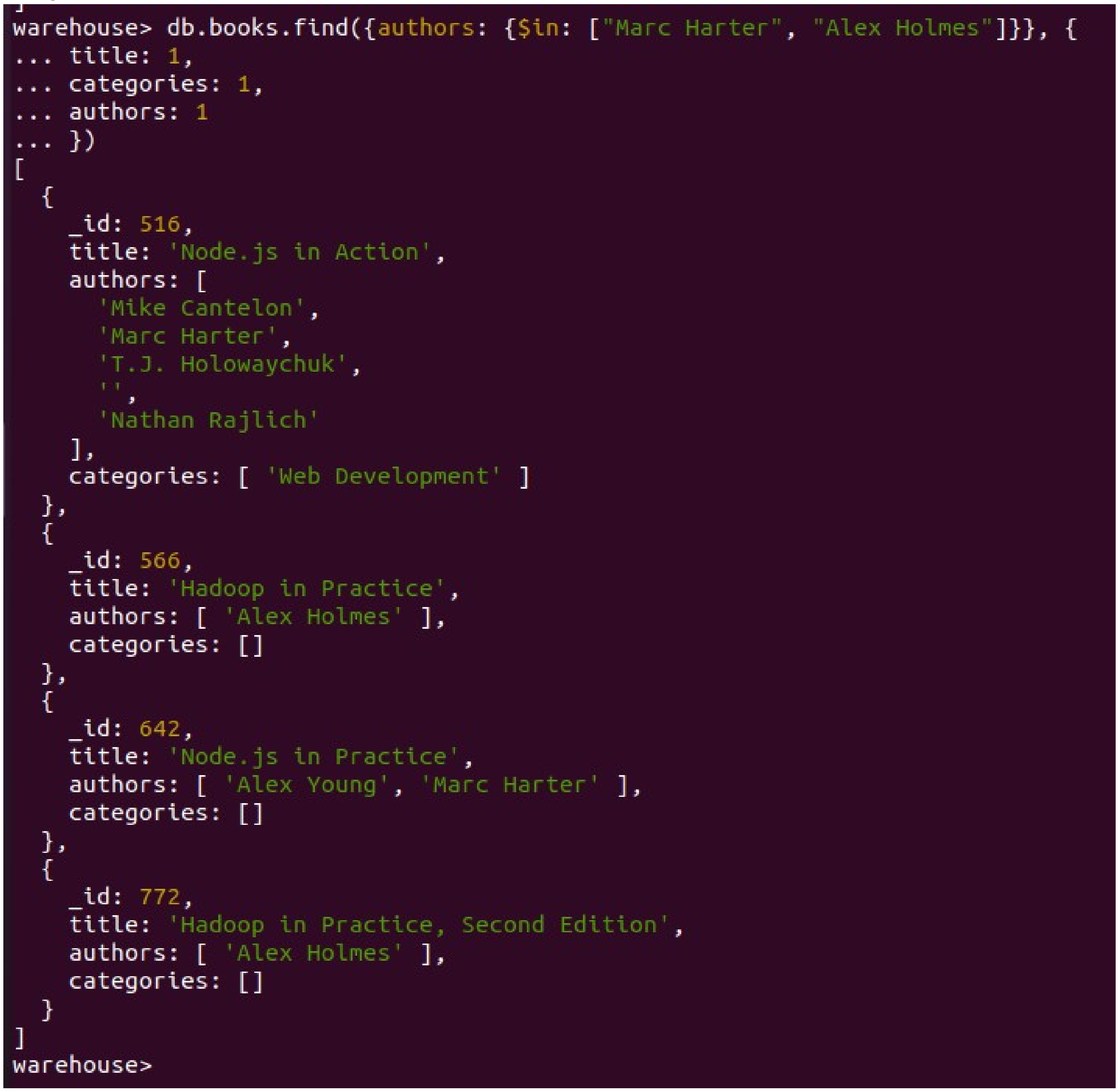
```
warehouse> db.books.find({categories: {$in: ["Python", "PHP"]}},{title: 1, categories: 1})
[
  {
    _id: 45,
    title: 'The Quick Python Book, Second Edition',
    categories: [ 'Python' ]
  },
  { _id: 37, title: 'Hello! Python', categories: [ 'Python' ] },
  {
    _id: 129,
    title: 'Python and Tkinter Programming',
    categories: [ 'Python' ]
  },
  {
    _id: 136,
    title: 'The Quick Python Book',
    categories: [ 'Python' ]
  },
  { _id: 263, title: 'wxPython in Action', categories: [ 'Python' ] },
  { _id: 267, title: 'PHP in Action', categories: [ 'PHP' ] },
  {
    _id: 278,
    title: 'Hello World!',
    categories: [ 'Programming', 'Python' ]
  }
]
```

Find the top 5 Python books with the most pages and print their titles, categories and page counts.



```
warehouse> db.books.find({categories: "Python"}, {title: 1, categories: 1, pageCount: 1}).sort({pageCount: -1}).limit(5)
[
  {
    _id: 129,
    title: 'Python and Tkinter Programming',
    pageCount: 688,
    categories: [ 'Python' ]
  },
  {
    _id: 263,
    title: 'wxPython in Action',
    pageCount: 620,
    categories: [ 'Python' ]
  },
  {
    _id: 136,
    title: 'The Quick Python Book',
    pageCount: 444,
    categories: [ 'Python' ]
  },
  {
    _id: 278,
    title: 'Hello World!',
    pageCount: 432,
    categories: [ 'Programming', 'Python' ]
  },
  {
    _id: 45,
    title: 'The Quick Python Book, Second Edition',
    pageCount: 360,
    categories: [ 'Python' ]
  }
]
warehouse>
```

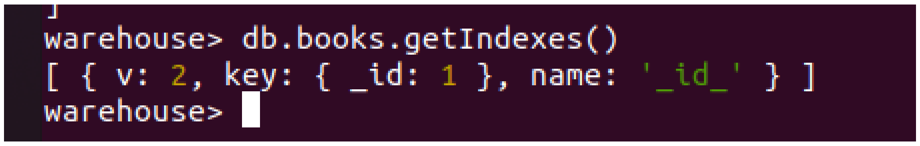
Find the books that have as author either "Marc Harter" or "Alex Holmes" and print their titles, authors and categories.



```
warehouse> db.books.find({authors: {$in: ["Marc Harter", "Alex Holmes"]}}, {
... title: 1,
... categories: 1,
... authors: 1
... })
[
  {
    _id: 516,
    title: 'Node.js in Action',
    authors: [
      'Mike Cantelon',
      'Marc Harter',
      'T.J. Holowaychuk',
      '',
      'Nathan Rajlich'
    ],
    categories: [ 'Web Development' ]
  },
  {
    _id: 566,
    title: 'Hadoop in Practice',
    authors: [ 'Alex Holmes' ],
    categories: []
  },
  {
    _id: 642,
    title: 'Node.js in Practice',
    authors: [ 'Alex Young', 'Marc Harter' ],
    categories: []
  },
  {
    _id: 772,
    title: 'Hadoop in Practice, Second Edition',
    authors: [ 'Alex Holmes' ],
    categories: []
  }
]
warehouse>
```

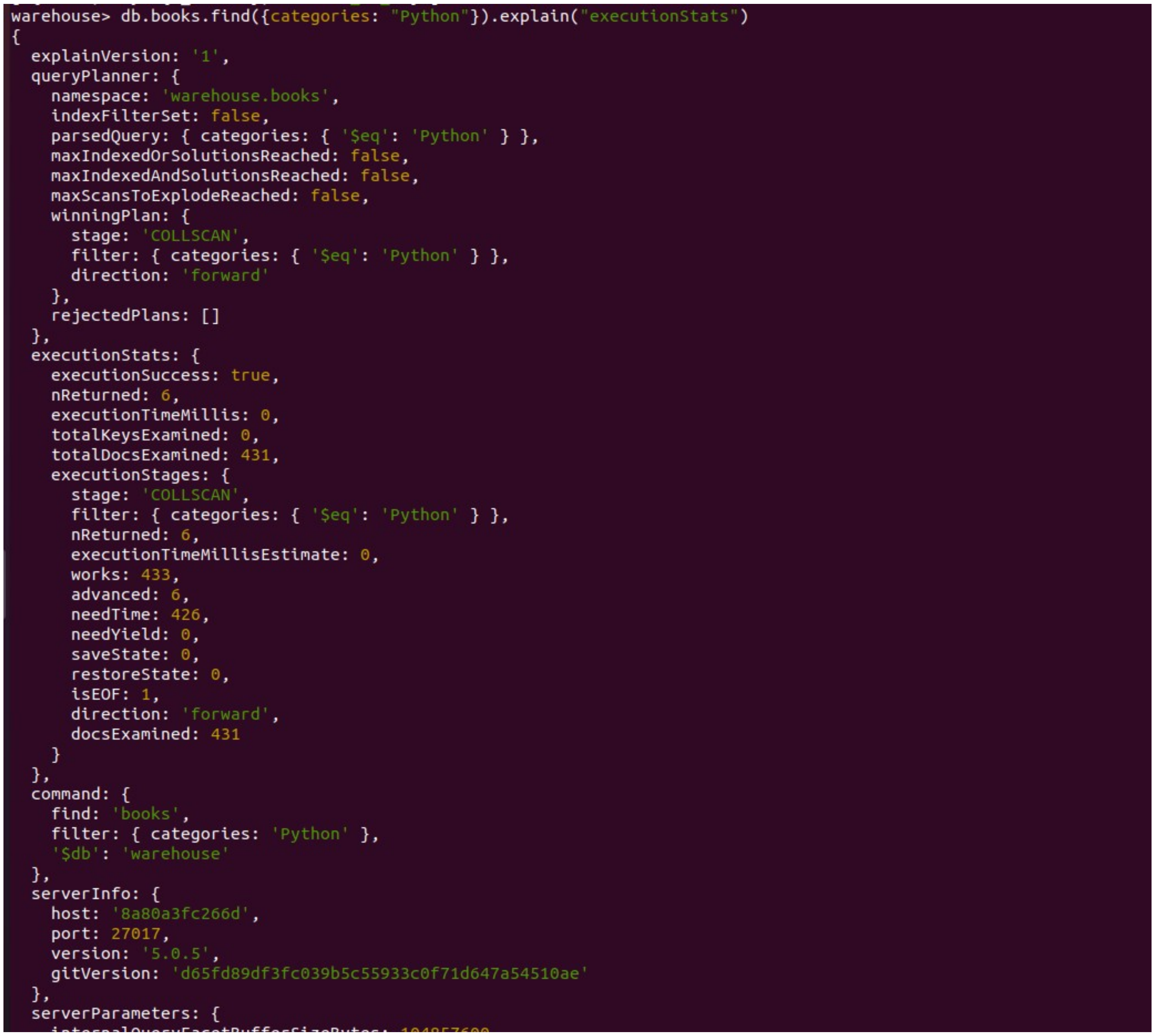
Indexing Documents

To see the indexes available for a collection run:

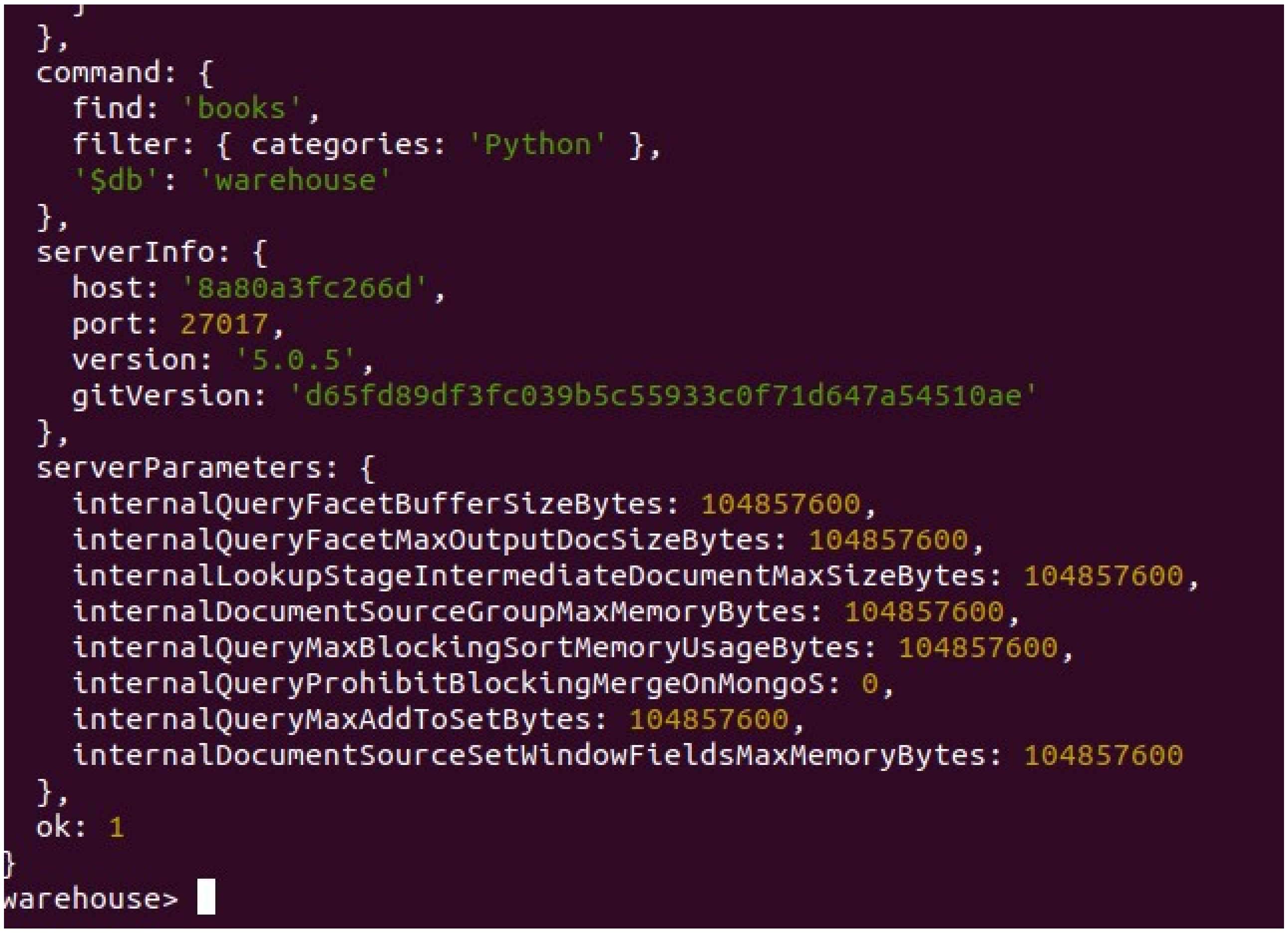


```
]
warehouse> db.books.getIndexes()
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
warehouse> 
```

To see how indexing helps query performance run the following query and examine its query plans and execution statistics using explain:

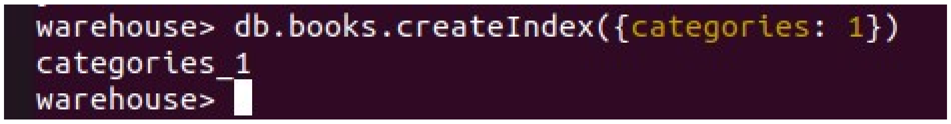


```
warehouse> db.books.find({categories: "Python"}).explain("executionStats")
{
  explainVersion: '1',
  queryPlanner: {
    namespace: 'warehouse.books',
    indexFilterSet: false,
    parsedQuery: { categories: { '$eq': 'Python' } },
    maxIndexedOrSolutionsReached: false,
    maxIndexedAndSolutionsReached: false,
    maxScansToExplodeReached: false,
    winningPlan: {
      stage: 'COLLSCAN',
      filter: { categories: { '$eq': 'Python' } },
      direction: 'forward'
    },
    rejectedPlans: []
  },
  executionStats: {
    executionSuccess: true,
    nReturned: 6,
    executionTimeMillis: 0,
    totalKeysExamined: 0,
    totalDocsExamined: 431,
    executionStages: {
      stage: 'COLLSCAN',
      filter: { categories: { '$eq': 'Python' } },
      nReturned: 6,
      executionTimeMillisEstimate: 0,
      works: 433,
      advanced: 6,
      needTime: 426,
      needYield: 0,
      saveState: 0,
      restoreState: 0,
      isEOF: 1,
      direction: 'forward',
      docsExamined: 431
    }
  },
  command: {
    find: 'books',
    filter: { categories: 'Python' },
    '$db': 'warehouse'
  },
  serverInfo: {
    host: '8a80a3fc266d',
    port: 27017,
    version: '5.0.5',
    gitVersion: 'd65fd89df3fc039b5c55933c0f71d647a54510ae'
  },
  serverParameters: {
```



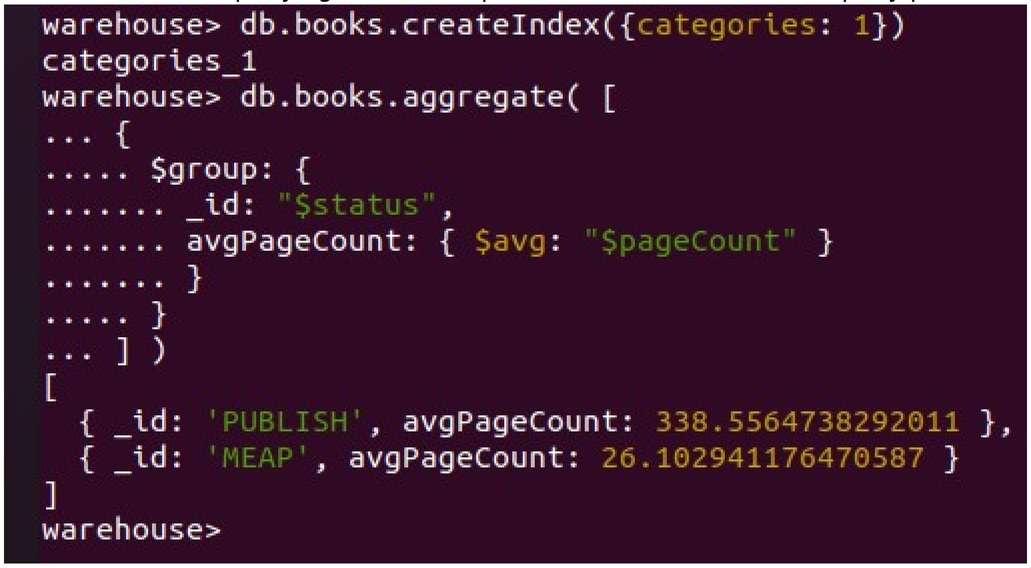
```
  },
  command: {
    find: 'books',
    filter: { categories: 'Python' },
    '$db': 'warehouse'
  },
  serverInfo: {
    host: '8a80a3fc266d',
    port: 27017,
    version: '5.0.5',
    gitVersion: 'd65fd89df3fc039b5c55933c0f71d647a54510ae'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600
  },
  ok: 1
}
warehouse>
```

Now add an index on the field categories:



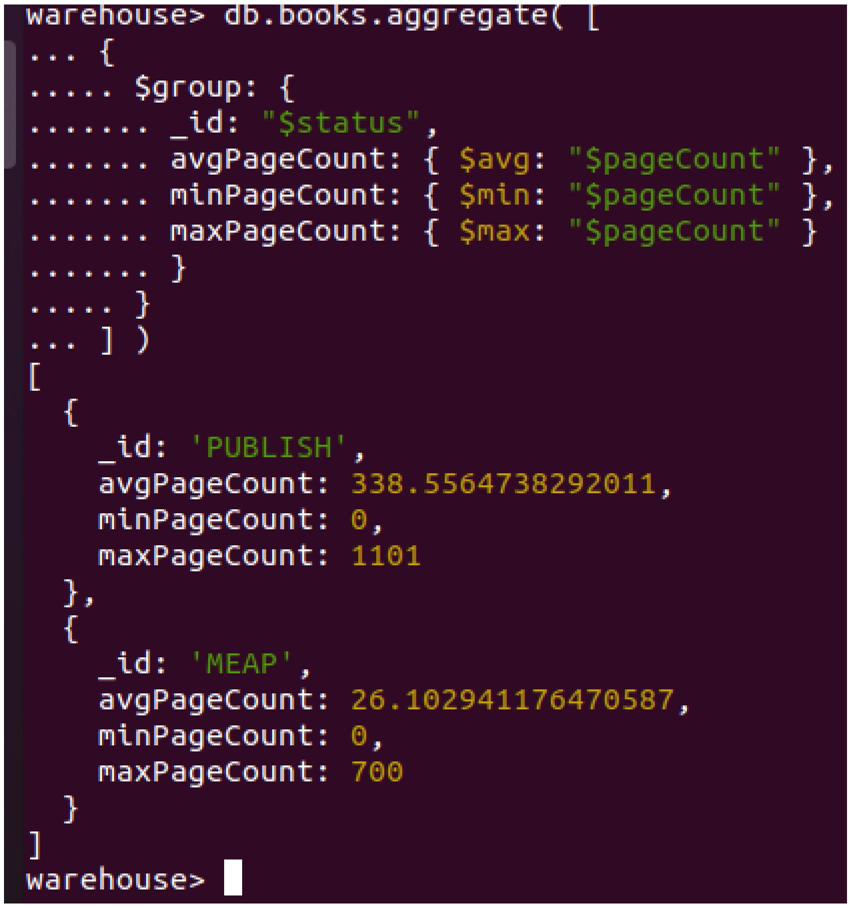
```
warehouse> db.books.createIndex({categories: 1})
categories_1
warehouse>
```

Now run the same query again and compare the execution stats and query plans followed.



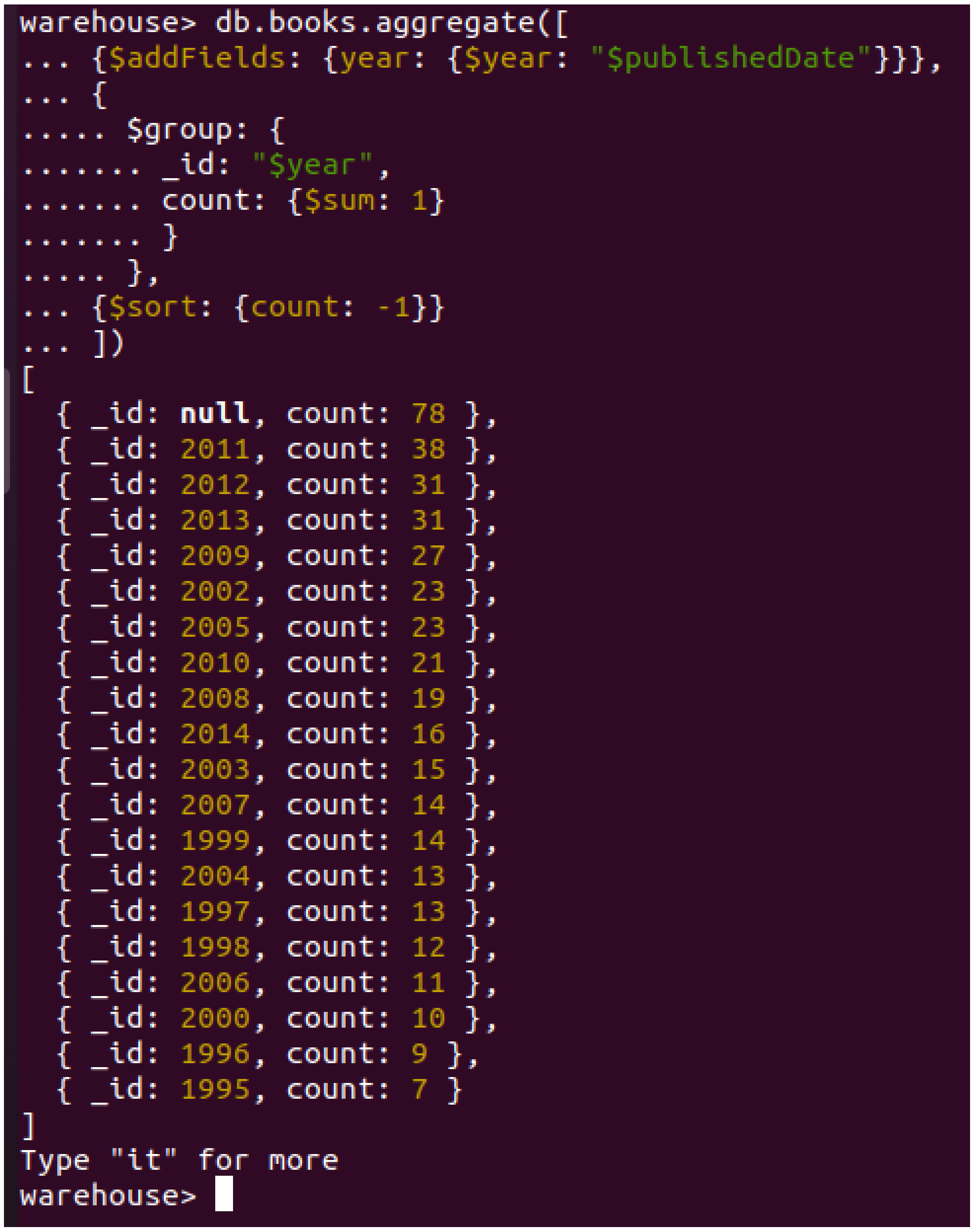
```
warehouse> db.books.createIndex({categories: 1})
categories_1
warehouse> db.books.aggregate( [
... {
..... $group: {
....... _id: "$status",
....... avgPageCount: { $avg: "$pageCount" }
....... }
..... }
... ] )
[
  { _id: 'PUBLISH', avgPageCount: 338.5564738292011 },
  { _id: 'MEAP', avgPageCount: 26.102941176470587 }
]
warehouse>
```

Expand the example above in order to also compute the minimum and maximum number of pages.



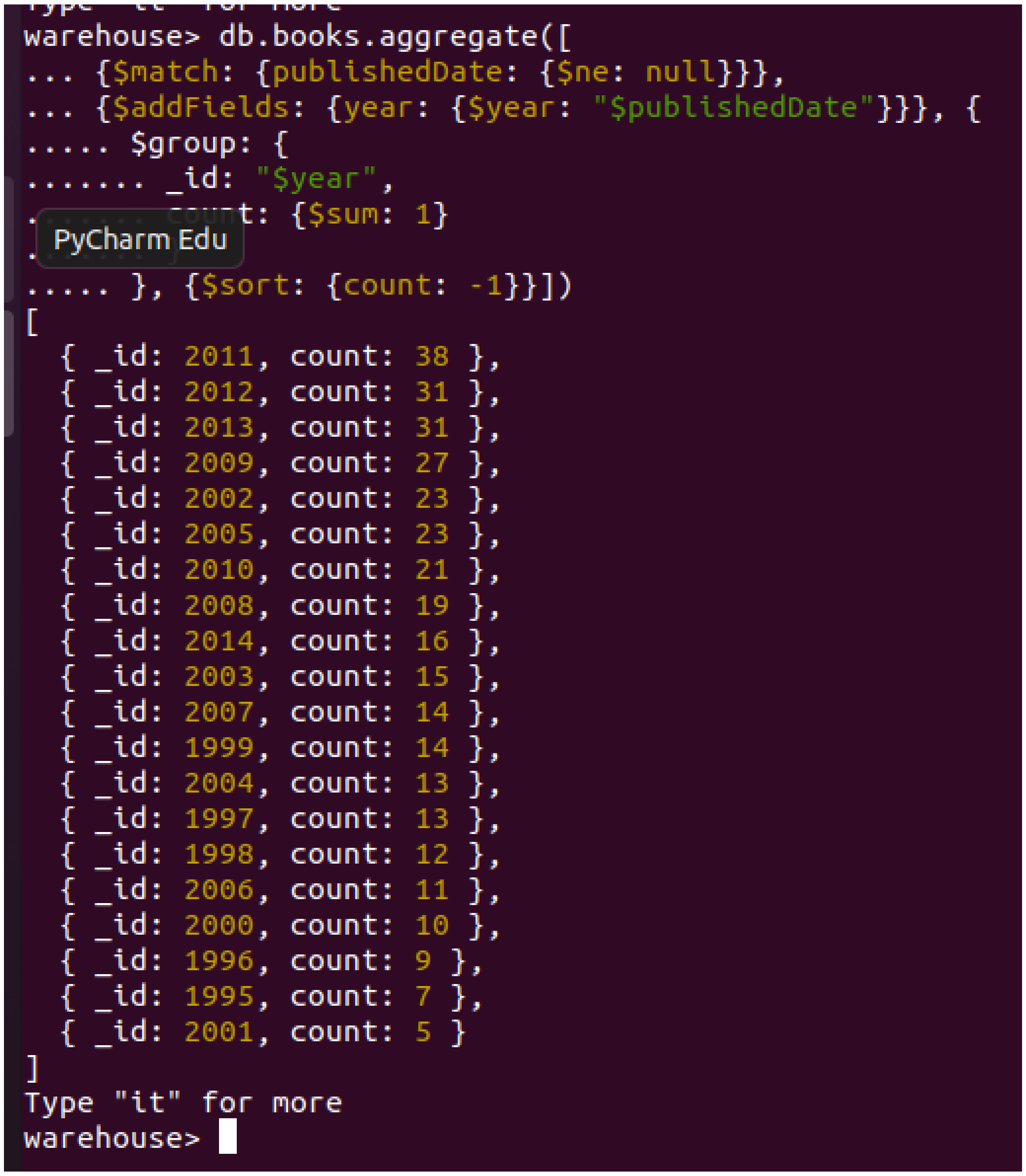
```
warehouse> db.books.aggregate( [
... {
..... $group: {
....... _id: "$status",
....... avgPageCount: { $avg: "$pageCount" },
....... minPageCount: { $min: "$pageCount" },
....... maxPageCount: { $max: "$pageCount" }
....... }
..... }
... ] )
[
  {
    _id: 'PUBLISH',
    avgPageCount: 338.5564738292011,
    minPageCount: 0,
    maxPageCount: 1101
  },
  {
    _id: 'MEAP',
    avgPageCount: 26.102941176470587,
    minPageCount: 0,
    maxPageCount: 700
  }
]
warehouse> 
```

Now we compute the number of books in the database per year. For this we can use the $year operator and add to each document a year field before the $group stage:



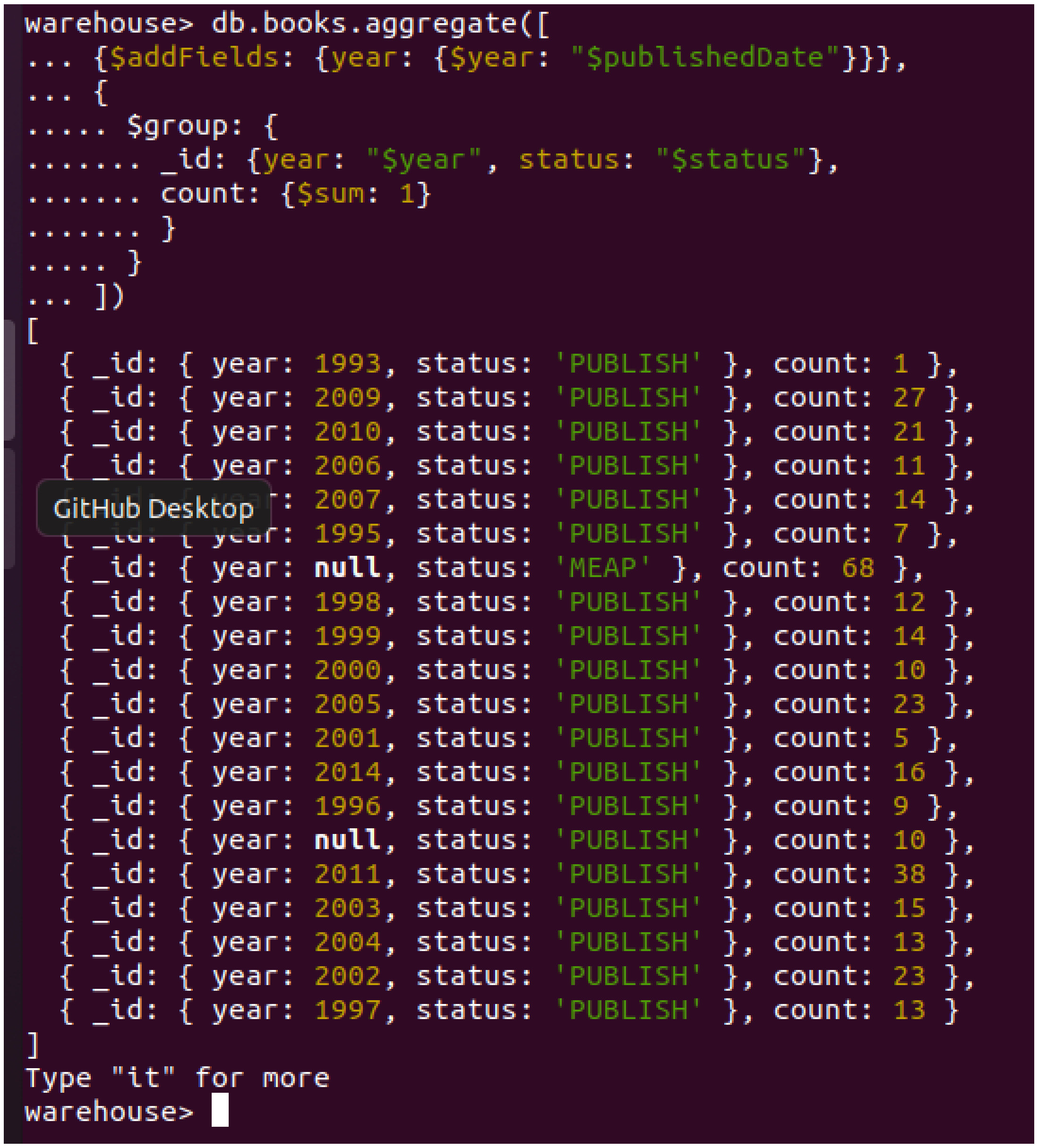
```
warehouse> db.books.aggregate([
... {$addFields: {year: {$year: "$publishedDate"}}},
... {
..... $group: {
....... _id: "$year",
....... count: {$sum: 1}
....... }
..... },
... {$sort: {count: -1}}
... ])
[
  { _id: null, count: 78 },
  { _id: 2011, count: 38 },
  { _id: 2012, count: 31 },
  { _id: 2013, count: 31 },
  { _id: 2009, count: 27 },
  { _id: 2002, count: 23 },
  { _id: 2005, count: 23 },
  { _id: 2010, count: 21 },
  { _id: 2008, count: 19 },
  { _id: 2014, count: 16 },
  { _id: 2003, count: 15 },
  { _id: 2007, count: 14 },
  { _id: 1999, count: 14 },
  { _id: 2004, count: 13 },
  { _id: 1997, count: 13 },
  { _id: 1998, count: 12 },
  { _id: 2006, count: 11 },
  { _id: 2000, count: 10 },
  { _id: 1996, count: 9 },
  { _id: 1995, count: 7 }
]
Type "it" for more
warehouse> 
```

To exclude books with no publication data available, we add a $match aggregation stage:



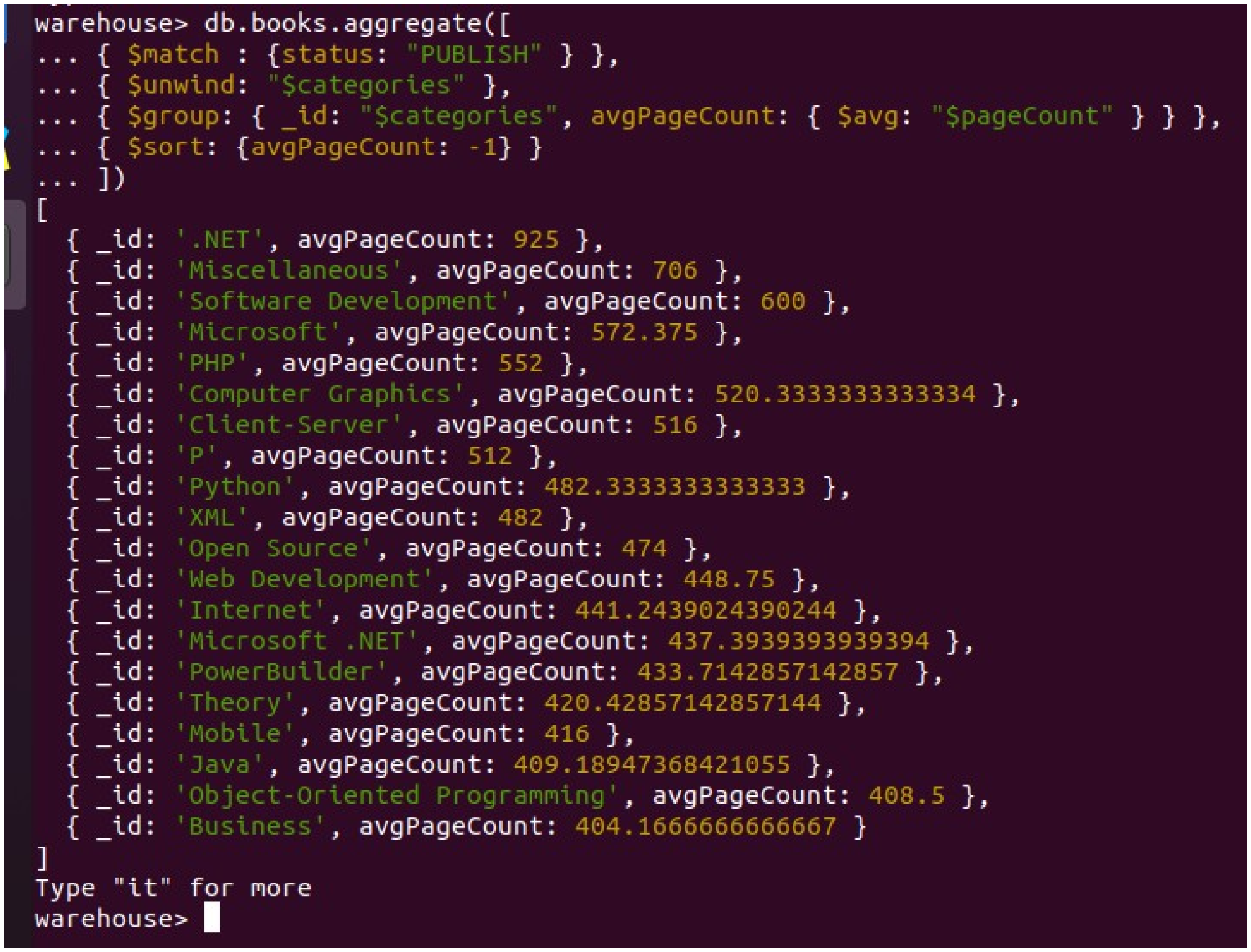
```
warehouse> db.books.aggregate([
... {$match: {publishedDate: {$ne: null}}},
... {$addFields: {year: {$year: "$publishedDate"}}}, {
..... $group: {
....... _id: "$year",
....... count: {$sum: 1}
....... }
..... }, {$sort: {count: -1}}])
[
  { _id: 2011, count: 38 },
  { _id: 2012, count: 31 },
  { _id: 2013, count: 31 },
  { _id: 2009, count: 27 },
  { _id: 2002, count: 23 },
  { _id: 2005, count: 23 },
  { _id: 2010, count: 21 },
  { _id: 2008, count: 19 },
  { _id: 2014, count: 16 },
  { _id: 2003, count: 15 },
  { _id: 2007, count: 14 },
  { _id: 1999, count: 14 },
  { _id: 2004, count: 13 },
  { _id: 1997, count: 13 },
  { _id: 1998, count: 12 },
  { _id: 2006, count: 11 },
  { _id: 2000, count: 10 },
  { _id: 1996, count: 9 },
  { _id: 1995, count: 7 },
  { _id: 2001, count: 5 }
]
Type "it" for more
warehouse> 
```

Expand the query above to find the number of books per year and status.



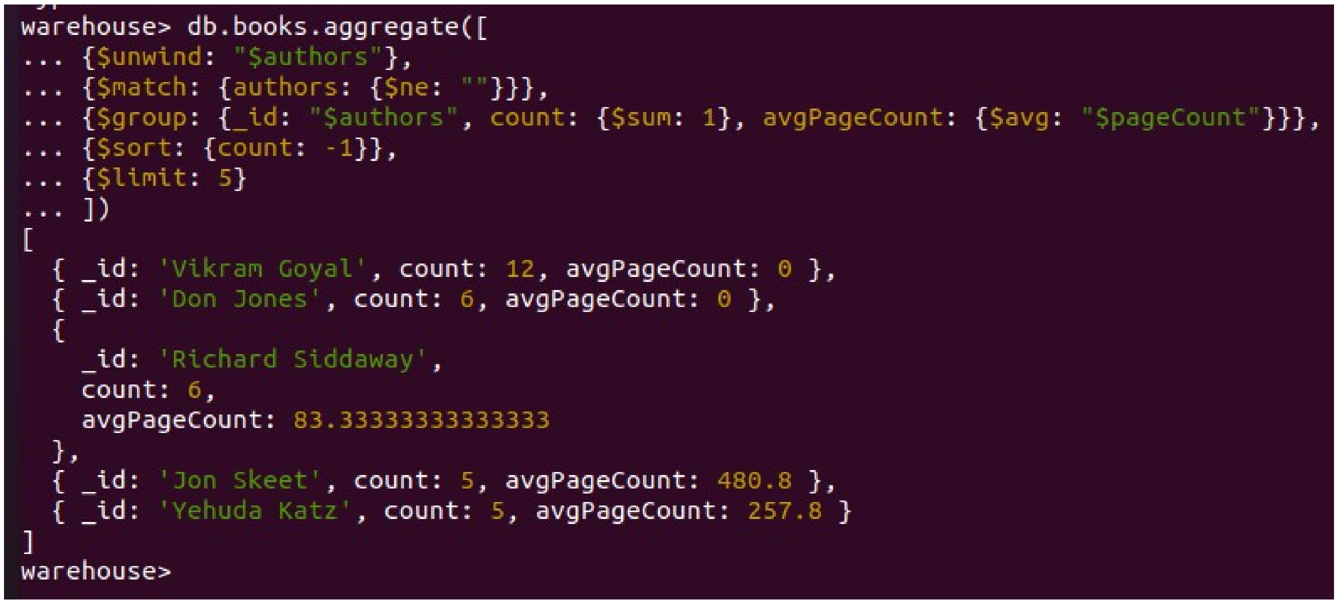
```
warehouse> db.books.aggregate([
... {$addFields: {year: {$year: "$publishedDate"}}},
... {
..... $group: {
....... _id: {year: "$year", status: "$status"},
....... count: {$sum: 1}
....... }
..... }
... ])
[
  { _id: { year: 1993, status: 'PUBLISH' }, count: 1 },
  { _id: { year: 2009, status: 'PUBLISH' }, count: 27 },
  { _id: { year: 2010, status: 'PUBLISH' }, count: 21 },
  { _id: { year: 2006, status: 'PUBLISH' }, count: 11 },
  { _id: { year: 2007, status: 'PUBLISH' }, count: 14 },
  { _id: { year: 1995, status: 'PUBLISH' }, count: 7 },
  { _id: { year: null, status: 'MEAP' }, count: 68 },
  { _id: { year: 1998, status: 'PUBLISH' }, count: 12 },
  { _id: { year: 1999, status: 'PUBLISH' }, count: 14 },
  { _id: { year: 2000, status: 'PUBLISH' }, count: 10 },
  { _id: { year: 2005, status: 'PUBLISH' }, count: 23 },
  { _id: { year: 2001, status: 'PUBLISH' }, count: 5 },
  { _id: { year: 2014, status: 'PUBLISH' }, count: 16 },
  { _id: { year: 1996, status: 'PUBLISH' }, count: 9 },
  { _id: { year: null, status: 'PUBLISH' }, count: 10 },
  { _id: { year: 2011, status: 'PUBLISH' }, count: 38 },
  { _id: { year: 2003, status: 'PUBLISH' }, count: 15 },
  { _id: { year: 2004, status: 'PUBLISH' }, count: 13 },
  { _id: { year: 2002, status: 'PUBLISH' }, count: 23 },
  { _id: { year: 1997, status: 'PUBLISH' }, count: 13 }
]
Type "it" for more
warehouse> 
```

In the following example, we want to find the average number of pages per book category for all books with status="PUBLISH", and sort the results by the average
page count. Remember that categories is an array field with possibly more than one values for every book.



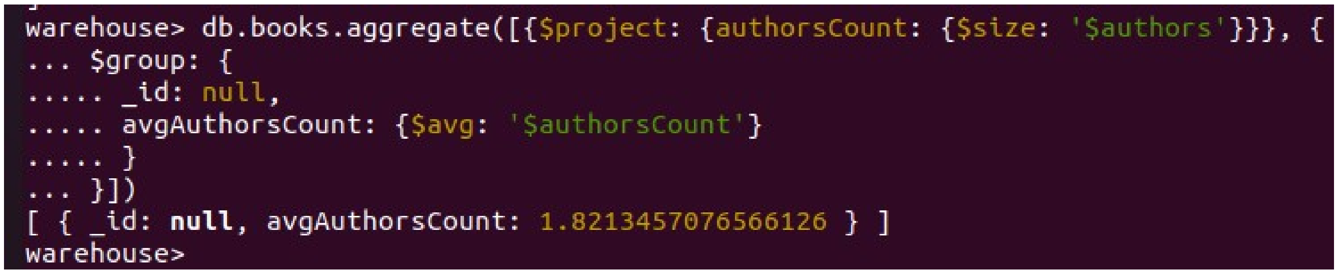
```
warehouse> db.books.aggregate([
... { $match : {status: "PUBLISH" } },
... { $unwind: "$categories" },
... { $group: { _id: "$categories", avgPageCount: { $avg: "$pageCount" } } },
... { $sort: {avgPageCount: -1} }
... ])
[
  { _id: '.NET', avgPageCount: 925 },
  { _id: 'Miscellaneous', avgPageCount: 706 },
  { _id: 'Software Development', avgPageCount: 600 },
  { _id: 'Microsoft', avgPageCount: 572.375 },
  { _id: 'PHP', avgPageCount: 552 },
  { _id: 'Computer Graphics', avgPageCount: 520.3333333333334 },
  { _id: 'Client-Server', avgPageCount: 516 },
  { _id: 'P', avgPageCount: 512 },
  { _id: 'Python', avgPageCount: 482.3333333333333 },
  { _id: 'XML', avgPageCount: 482 },
  { _id: 'Open Source', avgPageCount: 474 },
  { _id: 'Web Development', avgPageCount: 448.75 },
  { _id: 'Internet', avgPageCount: 441.2439024390244 },
  { _id: 'Microsoft .NET', avgPageCount: 437.3939393939394 },
  { _id: 'PowerBuilder', avgPageCount: 433.7142857142857 },
  { _id: 'Theory', avgPageCount: 420.42857142857144 },
  { _id: 'Mobile', avgPageCount: 416 },
  { _id: 'Java', avgPageCount: 409.18947368421055 },
  { _id: 'Object-Oriented Programming', avgPageCount: 408.5 },
  { _id: 'Business', avgPageCount: 404.1666666666667 }
]
Type "it" for more
warehouse>
```

Find the 5 authors with the most books, and print the number of books, as well as the average number of pages for each one.



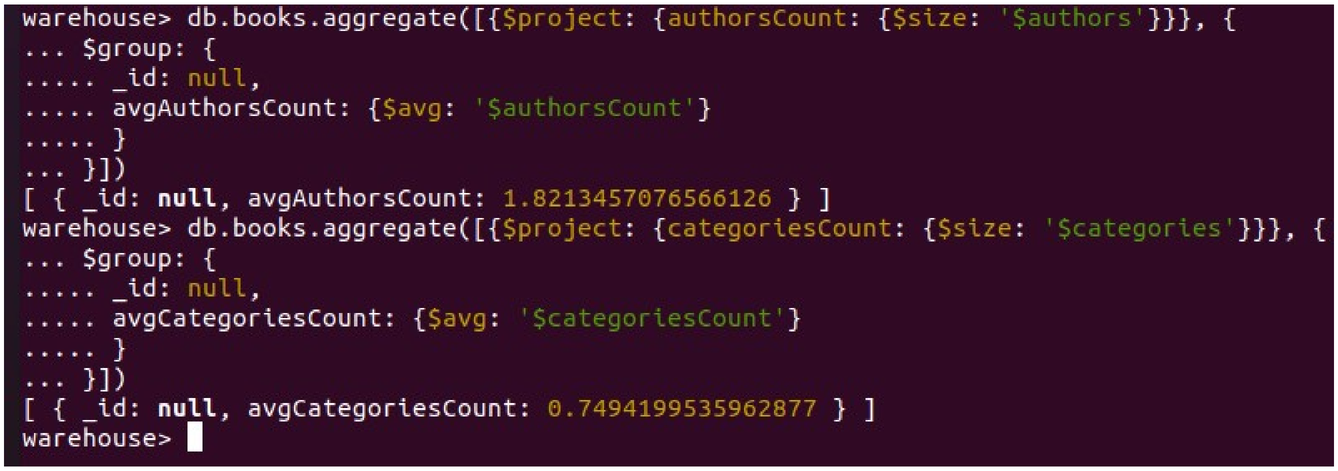
```
warehouse> db.books.aggregate([
... {$unwind: "$authors"},
... {$match: {authors: {$ne: ""}}},
... {$group: {_id: "$authors", count: {$sum: 1}, avgPageCount: {$avg: "$pageCount"}}},
... {$sort: {count: -1}},
... {$limit: 5}
... ])
[
  { _id: 'Vikram Goyal', count: 12, avgPageCount: 0 },
  { _id: 'Don Jones', count: 6, avgPageCount: 0 },
  {
    _id: 'Richard Siddaway',
    count: 6,
    avgPageCount: 83.33333333333333
  },
  { _id: 'Jon Skeet', count: 5, avgPageCount: 480.8 },
  { _id: 'Yehuda Katz', count: 5, avgPageCount: 257.8 }
]
warehouse>
```

Next, we need to find the average number of authors for all books. We use the $size operator which counts and returns the total number of items in an array.



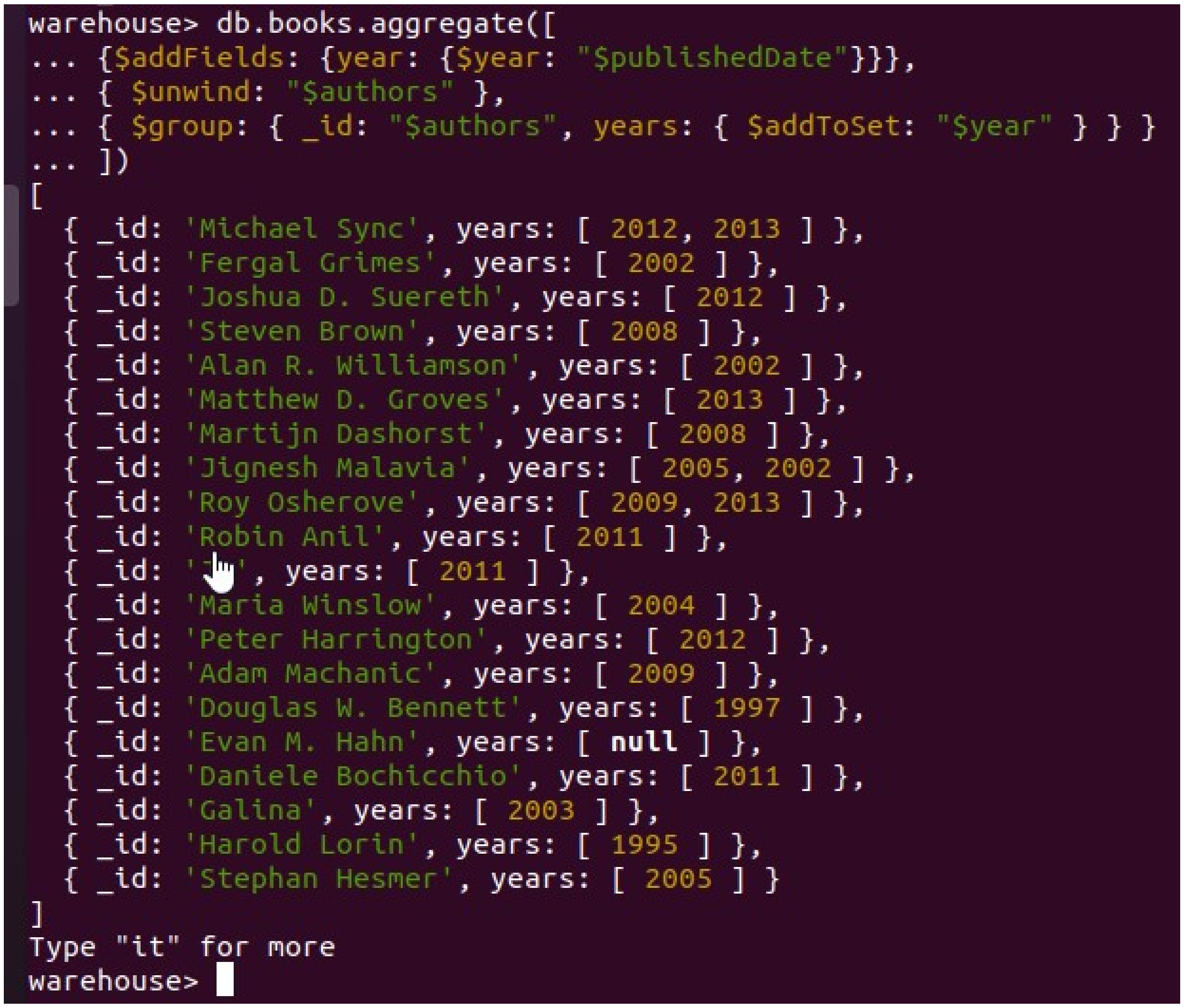
```
warehouse> db.books.aggregate([{$project: {authorsCount: {$size: '$authors'}}}, {
... $group: {
..... _id: null,
..... avgAuthorsCount: {$avg: '$authorsCount'}
..... }
... }])
[ { _id: null, avgAuthorsCount: 1.8213457076566126 } ]
warehouse>
```

In the same fashion, find the average number of categories for every book.



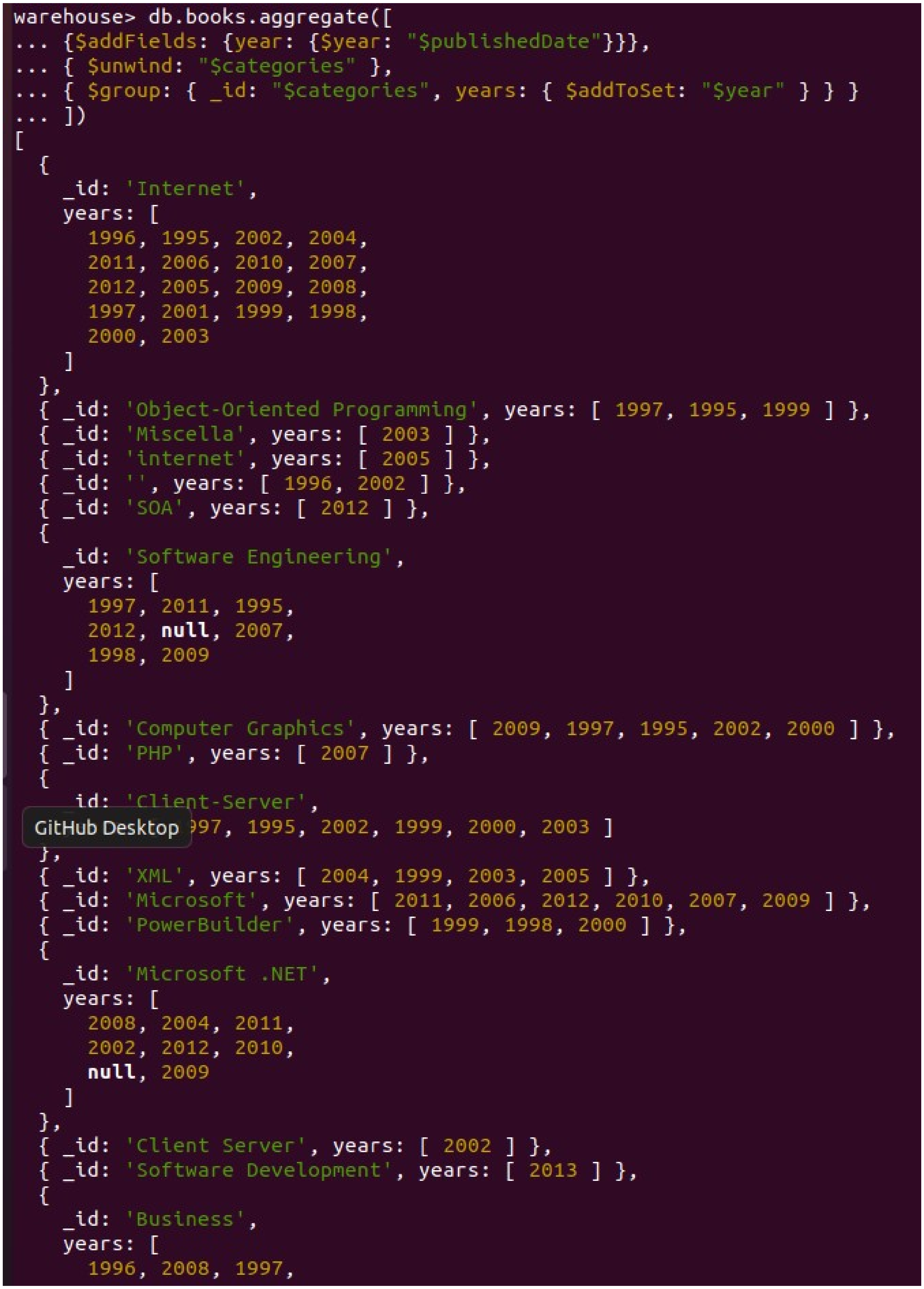
```
warehouse> db.books.aggregate([{$project: {authorsCount: {$size: '$authors'}}}, {
... $group: {
..... _id: null,
..... avgAuthorsCount: {$avg: '$authorsCount'}
..... }
... }])
[ { _id: null, avgAuthorsCount: 1.8213457076566126 } ]
warehouse> db.books.aggregate([{$project: {categoriesCount: {$size: '$categories'}}}, {
... $group: {
..... _id: null,
..... avgCategoriesCount: {$avg: '$categoriesCount'}
..... }
... }])
[ { _id: null, avgCategoriesCount: 0.7494199535962877 } ]
warehouse>
```

If we wish to find for every author the years that they published a book, we can run the following:

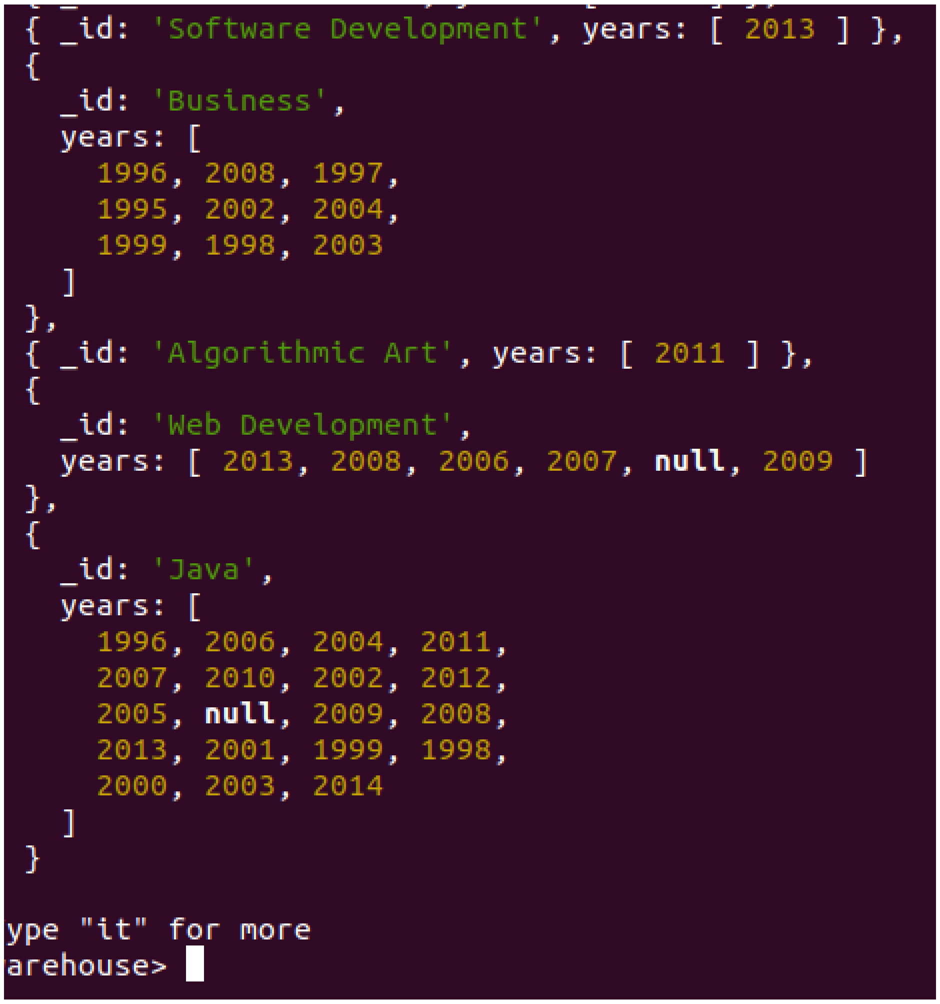


```
warehouse> db.books.aggregate([
... {$addFields: {year: {$year: "$publishedDate"}}},
... { $unwind: "$authors" },
... { $group: { _id: "$authors", years: { $addToSet: "$year" } } }
... ])
[
  { _id: 'Michael Sync', years: [ 2012, 2013 ] },
  { _id: 'Fergal Grimes', years: [ 2002 ] },
  { _id: 'Joshua D. Suereth', years: [ 2012 ] },
  { _id: 'Steven Brown', years: [ 2008 ] },
  { _id: 'Alan R. Williamson', years: [ 2002 ] },
  { _id: 'Matthew D. Groves', years: [ 2013 ] },
  { _id: 'Martijn Dashorst', years: [ 2008 ] },
  { _id: 'Jignesh Malavia', years: [ 2005, 2002 ] },
  { _id: 'Roy Osherove', years: [ 2009, 2013 ] },
  { _id: 'Robin Anil', years: [ 2011 ] },
  { _id: '[illegible]', years: [ 2011 ] },
  { _id: 'Maria Winslow', years: [ 2004 ] },
  { _id: 'Peter Harrington', years: [ 2012 ] },
  { _id: 'Adam Machanic', years: [ 2009 ] },
  { _id: 'Douglas W. Bennett', years: [ 1997 ] },
  { _id: 'Evan M. Hahn', years: [ null ] },
  { _id: 'Daniele Bochicchio', years: [ 2011 ] },
  { _id: 'Galina', years: [ 2003 ] },
  { _id: 'Harold Lorin', years: [ 1995 ] },
  { _id: 'Stephan Hesmer', years: [ 2005 ] }
]
Type "it" for more
warehouse> 
```

Now, find all the years that there were publications for every category of book:



```
warehouse> db.books.aggregate([
... {$addFields: {year: {$year: "$publishedDate"}}},
... { $unwind: "$categories" },
... { $group: { _id: "$categories", years: { $addToSet: "$year" } } }
... ])
[
  {
    _id: 'Internet',
    years: [
      1996, 1995, 2002, 2004,
      2011, 2006, 2010, 2007,
      2012, 2005, 2009, 2008,
      1997, 2001, 1999, 1998,
      2000, 2003
    ]
  },
  { _id: 'Object-Oriented Programming', years: [ 1997, 1995, 1999 ] },
  { _id: 'Miscella', years: [ 2003 ] },
  { _id: 'internet', years: [ 2005 ] },
  { _id: '', years: [ 1996, 2002 ] },
  { _id: 'SOA', years: [ 2012 ] },
  {
    _id: 'Software Engineering',
    years: [
      1997, 2011, 1995,
      2012, null, 2007,
      1998, 2009
    ]
  },
  { _id: 'Computer Graphics', years: [ 2009, 1997, 1995, 2002, 2000 ] },
  { _id: 'PHP', years: [ 2007 ] },
  {
    _id: 'Client-Server',
              997, 1995, 2002, 1999, 2000, 2003 ]
  },
  { _id: 'XML', years: [ 2004, 1999, 2003, 2005 ] },
  { _id: 'Microsoft', years: [ 2011, 2006, 2012, 2010, 2007, 2009 ] },
  { _id: 'PowerBuilder', years: [ 1999, 1998, 2000 ] },
  {
    _id: 'Microsoft .NET',
    years: [
      2008, 2004, 2011,
      2002, 2012, 2010,
      null, 2009
    ]
  },
  { _id: 'Client Server', years: [ 2002 ] },
  { _id: 'Software Development', years: [ 2013 ] },
  {
    _id: 'Business',
    years: [
      1996, 2008, 1997,
```



```
 { _id: 'Software Development', years: [ 2013 ] },
 {
   _id: 'Business',
   years: [
     1996, 2008, 1997,
     1995, 2002, 2004,
     1999, 1998, 2003
   ]
 },
 { _id: 'Algorithmic Art', years: [ 2011 ] },
 {
   _id: 'Web Development',
   years: [ 2013, 2008, 2006, 2007, null, 2009 ]
 },
 {
   _id: 'Java',
   years: [
     1996, 2006, 2004, 2011,
     2007, 2010, 2002, 2012,
     2005, null, 2009, 2008,
     2013, 2001, 1999, 1998,
     2000, 2003, 2014
   ]
 }

ype "it" for more
arehouse> 
```