MONGODB ASSIGNMENT 1

use('mongo\_practice');

db.movies.insertMany([

{

title : "Fight Club",

writer : "Chuck Palahniu",

year : 1999,

actors : [

"Brad Pitt",

"Edward Norton"

]

},

{

title : "Pulp Fiction",

writer : "Quentin Tarantino",

year : 1994,

actors : [

"John Travolta",

"Uma Thurman"

]

},

{

title : "Inglorious Basterds",

writer : "Quentin Tarantino",

year : 2009,

actors : [

"Brad Pitt",

"Diane Kruger",

"Eli Roth"

]

},

{

title : "The Hobbit: An Unexpected Journey",

writer : "J.R.R. Tolkein",

year : 2012,

franchise : "The Hobbit"

},

{

title : "The Hobbit: The Desolation of Smaug",

writer : "J.R.R. Tolkein",

year : 2013,

franchise : "The Hobbit"

},

{

title : "The Hobbit: The Battle of the Five Armies",

writer : "J.R.R. Tolkein",

year : 2012,

franchise : "The Hobbit",

synopsis : "Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness."

},

{

title : "Pee Wee Herman's Big Adventure"

},

{

title : "Avatar"

}

]

)

db.movies.find()

db.movies.find({writer : "Quentin Tarantino"})

db.movies.find({actors : "Brad Pitt"})

db.movies.find({franchise : "The Hobbit"})

db.movies.find({year: {$lt: 2000}})

db.movies.find({$and: [{year: {$gt: 1990}}, {year: {$lt: 2000}}]})

s

db.movies.find({$or: [{year: {$gt: 2010}}, {year: {$lt: 2000}}]})

db.movies.updateOne({title : "The Hobbit: An Unexpected Journey"},{/\*\*

\* field: The field name

\* expression: The expression.

\*/

$set: {

synopsis: "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."

}})

db.movies.find({})

db.movies.updateOne({title : "The Hobbit: The Desolation of Smaug"},{/\*\*

\* field: The field name

\* expression: The expression.

\*/

$set: {

synopsis: "The dwarves, along with Bilbo Baggins and Gandalf the Grey, continue their quest to reclaim Erebor, their homeland, from Smaug. Bilbo Baggins is in possession of a mysterious and magical ring."

}})

db.movies.updateOne({title: "Pulp Fiction"}, {/\*\*

\* field: The field name

\* expression: The expression.

\*/

$set: {

actor: [

"Samuel L. Jackson"

]

}})

db.movies.update({title: "Pulp Fiction"}, {$unset: {

actor: ""

}})

db.movies.update({title: "Pulp Fiction"}, {/\*\*

\* field: The field name

\* expression: The expression.

\*/

$push: {

actors:

"Samuel L. Jackson"

}})

db.movies.createIndex({

synopsis: "text"

})

db.movies.find({$text: {/\*\*

\* index: the name of the Search index.

\* text: Analyzed search, with required fields of query and path, the analyzed field(s) to search.

\* term: Un-analyzed search.

\* compound: Combines ops.

\* span: Find in text field regions.

\* exists: Test for presence of a field.

\* near: Find near number or date.

\* range: Find in numeric or date range.

\*/

$search: "Bilbo"

}})

db.movies.find({$text: {/\*\*

\* index: the name of the Search index.

\* text: Analyzed search, with required fields of query and path, the analyzed field(s) to search.

\* term: Un-analyzed search.

\* compound: Combines ops.

\* span: Find in text field regions.

\* exists: Test for presence of a field.

\* near: Find near number or date.

\* range: Find in numeric or date range.

\*/

$search: "Gandalf"

}})

db.movies.find({$text: {/\*\*

\* index: the name of the Search index.

\* text: Analyzed search, with required fields of query and path, the analyzed field(s) to search.

\* term: Un-analyzed search.

\* compound: Combines ops.

\* span: Find in text field regions.

\* exists: Test for presence of a field.

\* near: Find near number or date.

\* range: Find in numeric or date range.

\*/

$search: "Bilbo -Gandalf"

}})

db.movies.find({$text: {/\*\*

\* index: the name of the Search index.

\* text: Analyzed search, with required fields of query and path, the analyzed field(s) to search.

\* term: Un-analyzed search.

\* compound: Combines ops.

\* span: Find in text field regions.

\* exists: Test for presence of a field.

\* near: Find near number or date.

\* range: Find in numeric or date range.

\*/

$search: "dwarves hobbit"

}})

db.movies.find({$text : {/\*\*

\* index: the name of the Search index.

\* text: Analyzed search, with required fields of query and path, the analyzed field(s) to search.

\* term: Un-analyzed search.

\* compound: Combines ops.

\* span: Find in text field regions.

\* exists: Test for presence of a field.

\* near: Find near number or date.

\* range: Find in numeric or date range.

\*/

$search: "\"gold\" \"dragon\""

}})

db.movies.deleteMany({title : "Pee Wee Herman's Big Adventure"})

db.movies.find({})

db.movies.deleteMany({title : "Avatar"})

use('mongo\_practice');

db.createCollection('posts');

use('mongo\_practice');

db.users.insertMany(

[

{username : "GoodGuyGreg",

first\_name : "Good Guy",

last\_name : "Greg"

},

{

username : "ScumbagSteve",

full\_name :

{

first : "Scumbag",

last : "Steve"

}

}

]

)

db.posts.insertMany([

{

username : "GoodGuyGreg",

title : "Passes out at party",

body : "Wakes up early and cleans house"

},

{

username : "GoodGuyGreg",

title : "Steals your identity",

body : "Raises your credit score"

},

{

username : "GoodGuyGreg",

title : "Reports a bug in your code",

body : "Sends you a Pull Request"

},

{

username : "ScumbagSteve",

title : "Borrows something",

body : "Sells it"

},

{

username : "ScumbagSteve",

title : "Borrows everything",

body : "The end"

},

{

username : "ScumbagSteve",

title : "Forks your repo on github",

body : "Sets to private"

}

])

use('mongo\_practice')

db.users.find({})

db.posts.find({})

use('mongo\_practice')

db.createCollection('comments')

use('mongo\_practice')

db.comments.insertMany([

{

username : "GoodGuyGreg",

comment : "Hope you got a good deal!",

post : ["60a75c97149f47319e2dae13"]

},

{

username : "GoodGuyGreg",

comment : "What's mine is yours!",

post : ["60a75c97149f47319e2dae14"]

},

{

username : "GoodGuyGreg",

comment : "Don't violate the licensing agreement!",

post : ["60a75c97149f47319e2dae15"]

},

{

username : "ScumbagSteve",

comment : "It still isn't clean",

post : ["60a75c97149f47319e2dae10"]

},

{

username : "ScumbagSteve",

comment : "Denied your PR cause I found a hack",

post : ["60a75c97149f47319e2dae12"]

}

])

use('mongo\_practice')

db.comments.find({})

use('mongo\_practice')

db.users.find({})

use('mongo\_practice')

db.posts.find({})

use('mongo\_practice')

db.posts.find({username: "GoodGuyGreg"})

use('mongo\_practice')

db.posts.find({username: "ScumbagSteve"})

use('mongo\_practice')

db.comments.find({})

use('mongo\_practice')

db.comments.find({username: "GoodGuyGreg"})

use('mongo\_practice')

db.comments.find({username: "ScumbagSteve"})

use('mongo\_practice')

db.comments.insertMany([

{

username : "GoodGuyGreg",

comment : "Hope you got a good deal!",

post : "60a75c97149f47319e2dae13"

},

{

username : "GoodGuyGreg",

comment : "What's mine is yours!",

post : "60a75c97149f47319e2dae14"

},

{

username : "GoodGuyGreg",

comment : "Don't violate the licensing agreement!",

post : "60a75c97149f47319e2dae15"

},

{

username : "ScumbagSteve",

comment : "It still isn't clean",

post : "60a75c97149f47319e2dae10"

},

{

username : "ScumbagSteve",

comment : "Denied your PR cause I found a hack",

post : "60a75c97149f47319e2dae12"

}

])

use('mongo\_practice')

db.comments.find({post: "60a75c97149f47319e2dae12"})

MONGODB ASSIGNMENT 2

use("population")

db.zipcodes.find({$and: [{city: "ATLANTA"}, {state: "GA"}]})

use("population")

db.zipcodes.aggregate([{$match: {$and: [{city: "ATLANTA"}, {state: "GA"}]}}])

use("population")

db.zipcodes.aggregate([

{

/\*\*

\* query: The query in MQL.

\*/

$match: {

city: "ATLANTA"

}

},

{/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$city",

ZipcodesCount: {

$sum : 1

}

}}

])

use("population")

db.zipcodes.aggregate([

{

/\*\*

\* query: The query in MQL.

\*/

$match: {

city: "ATLANTA"

}

},

{/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$city",

TotalPopulation: {

$sum : "$pop"

}

}}

])

use("population")

db.zipcodes.aggregate([

{/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$state",

TotalStatePopulation: {

$sum : "$pop"

}

}}

])

use("population")

db.zipcodes.aggregate([

{/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$state",

TotalStatePopulation: {

$sum : "$pop"

}

}},

{

$sort: {TotalStatePopulation: -1}

}

])

use("population")

db.zipcodes.aggregate([

{

$sort: {pop: -1}

}

])

use("population")

db.zipcodes.aggregate([

{/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$state",

TotalStatePopulation: {

$sum : "$pop"

}

}},

{

$sort: {TotalStatePopulation: -1}

},

{

/\*\*

\* Provide the number of documents to limit.

\*/

$limit: 3

}

])

use('population')

db.zipcodes.aggregate([

{

/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$city",

TotalCityPopulation: {

$sum: "$pop"

}

}

}

])

use('population')

db.zipcodes.aggregate([

{

/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$city",

TotalCityPopulation: {

$sum: "$pop"

}

}

},

{

$sort: {

TotalCityPopulation: -1

}

}

])

use('population')

db.zipcodes.aggregate([

{

/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: "$city",

TotalCityPopulation: {

$sum: "$pop"

}

}

},

{

$sort: {

TotalCityPopulation: -1

}

},

{

/\*\*

\* Provide the number of documents to limit.

\*/

$limit: 3

}

])

use("population")

db.zipcodes.aggregate([

{

/\*\*

\* query: The query in MQL.

\*/

$match: {

state: "TX"

}

},

{

/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: {city: "$city", state: "$state"},

TotalPopulationofCityinTexas: {

$sum: "$pop"

}

}

},

{

$sort: {

TotalPopulationofCityinTexas: -1

}

},

{

/\*\*

\* Provide the number of documents to limit.

\*/

$limit: 3

}

])

use("population")

db.zipcodes.aggregate(

[

{

/\*\*

\* \_id: The id of the group.

\* fieldN: The first field name.

\*/

$group: {

\_id: {city: "$city", state: "$state"},

TotalPopulationofCityinState: {

$avg: "$pop"

}

}

},

{

/\*\*

\* Provide any number of field/order pairs.

\*/

$sort: {

TotalPopulationofCityinState: -1

}

},

{

/\*\*

\* Provide the number of documents to limit.

\*/

$limit: 3

}

]

)

MONGODB ASSIGNMENT 3

use('restaurants')

db.addresses.find({})

use('restaurants')

db.addresses.find({},{restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({},{restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, \_id: 0})

use('restaurants')

db.addresses.find({},{restaurant\_id: 1, name: 1, borough: 1, cuisine: 1, "address.zipcode" : 1, \_id: 0})

use('restaurants')

db.addresses.find({borough: "Bronx"}).limit(5)

use('restaurants')

db.addresses.find({borough: "Bronx"})

use('restaurants')

db.addresses.find({borough: "Bronx"}).skip(5).limit(5)

use('restaurants')

db.addresses.find({"grades.score" : {$gt: 90}})

use('restaurants')

db.addresses.find({"grades":{$elemMatch:{score:{$gt:80, $lt:100}}}})

use('restaurants')

db.addresses.find({"address.coord.0": {$lt: -95.754168}})

use('restaurants')

db.addresses.find({$and: [{"cuisine": {$ne: "American "}}, {"grades":{$elemMatch:{score:{$gt:70}}}}, {"address.coord.0": {$lt: -65.754168}}]})

use('restaurants')

db.addresses.find({$and: [{"cuisine": {$ne: "American "}}, {"grades":{$elemMatch:{score:{$gt:70}}}}, {"address.coord.0": {$lt: -65.754168}}]})

use('restaurants')

db.addresses.find({$and: [{"cuisine": {$ne: "American "}}, {"grades.grade": "A"}, {"borough": {$ne: "Brooklyn"}}]}).sort({cuisine:-1})

use('restaurants')

db.addresses.find({name: {$regex: /^Wil/}},{restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({name: {$regex: /ces$/}},{restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({name: {$regex: /Reg/}},{restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({$and: [{borough: "Bronx"},{$or: [{cuisine:"American "},{cuisine: "Chinese"}]}]})

use('restaurants')

db.addresses.find({$or: [{borough: "Staten Island"}, {borough: "Queens"},{borough: "Bronxor Brooklyn"}]}, {restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({borough:{$not: {$in: ["Staten Island", "Queens", "Bronxor Brooklyn"]}}}, {restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({"grades.score": {$not: {$gt: 10}}}, {restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({$and: [{cuisine: {$not: {$in: ["American ", "Chinese"]}}}, {name: {$regex: /^Wil/}}]}, {restaurant\_id: 1, name: 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({"grades": {$elemMatch: {score: 11, "date": ISODate("2014-08-11T00:00:00Z"), grade: "A"}}}, {restaurant\_id: 1, name: 1, grades: 1})

use('restaurants')

db.addresses.find({"grades.1": {"date": ISODate("2014-08-11T00:00:00Z"), "grade": "A", score: 9}}, {restaurant\_id: 1, name: 1, grades: 1})

use('restaurants')

db.addresses.find({$and: [{"grades.1.grade":"A"}, {"grades.1.score": 9}, {"grades.1.date": ISODate("2014-08-11T00:00:00Z")}]},{\_id:0, restaurant\_id:1, name:1, grades:1}).pretty()

// {restaurant\_id: 1, name: 1, address: 1}

use('restaurants')

db.addresses.find({$and: [{"address.coord.1": {$gt: 42}}, {"address.coord.1": {$lte: 52}}]}, {restaurant\_id: 1, name: 1, address: 1}).pretty()

use('restaurants')

db.addresses.find({},{name:1}).sort({name: -1})

use('restaurants')

db.addresses.find({},{name:1}).sort({name: 1})

use('restaurants')

db.addresses.find({},{cuisine:1, borough:1}).sort({cuisine: 1}).sort({borough:1})

use('restaurants')

db.addresses.find({}).count()

use('restaurants')

db.addresses.count({$or: [{"address.street": {$exists: true}}, {"address.street": {$not: {$eq: null}}}]})

use('restaurants')

db.addresses.find({"address.street": {$regex: /Street/}})

use('restaurants')

db.addresses.find({$and: [{"address.coord.0": {$type: 1}}, {"address.coord.1": {$type: 1}}]}).pretty()

use('restaurants')

db.addresses.find({"grades": {$elemMatch: {score: {$mod: [7,0]}}}},{restaurant\_id: 1, name: 1, grade: 1}).pretty()

use('restaurants')

db.addresses.find({name: {$regex: /mon/}},{restaurant\_id: 1, "address.coord": 1, borough: 1, cuisine: 1})

use('restaurants')

db.addresses.find({name: {$regex: /^Mad/}},{restaurant\_id: 1, "address.coord": 1, borough: 1, cuisine: 1, name:1})