Hochschule RheinMain

# MONGO DB

## Ein Beispiel für eine documentstored DB

Prof. Dr. Eva-Maria Iwer
Mai Juni 2022

Hochschule RheinMain

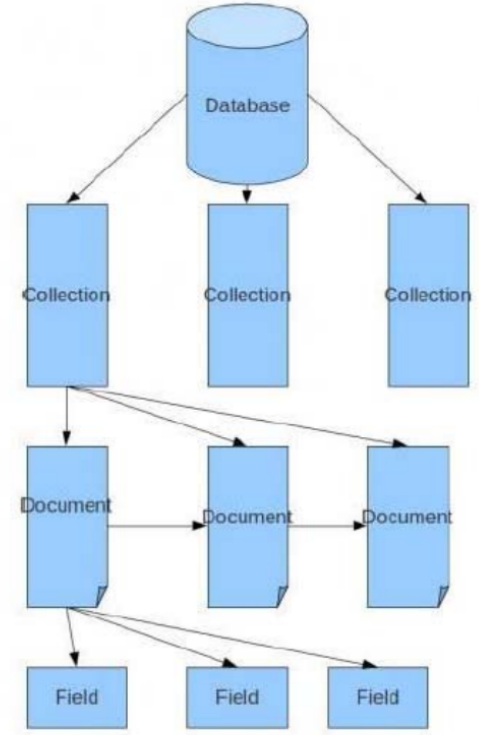
# MONGO DB

Die Theory

# MongoDB

mongoDB

- zunehmend Verbreitung findender Dokumenten-Store
  - open source-Version verfügbar
  - JSON-Dokumente, gespeichert als BSON (Binary JSON)
- DB besteht aus Kollektionen von Dokumenten
- einfache Anfragesprache
  - Indexierung von Attributen möglich
  - Map/Reduce-Unterstützung
- Skalierbarkeit und Fehlertoleranz
  - Skalierbarkeit durch horizontale Partitionierung der Dokumentenkollektionen unter vielen Knoten ("Sharding")
  - automatische Replikation mit Konsistenzwahrung
- kein ACID, z.B bzgl Synchronisation
  - Änderungen nur bzgl einzelner Dokumente atomar



# Beispiel: relational vs. dokumentenorientiert

**RDB:**

STUDENTS

| sno | name |
|---|---|
| 1 | Peter |
| 2 | Tom |

POSTS

| postID | topic | pdate |
|---|---|---|
| p1 | NoSQL | 01-09-2012 |
| p2 | RelationalDB | 02-09-2012 |

COMMENTS

| commentID | postID | sno | content | cdate |
|---|---|---|---|---|
| c1 | p1 | 1 | Excellent | 02-09-2012 |
| c2 | p1 | 2 | I do not understand. | 03-09-2012 |
| c3 | p1 | 1 | This is a good idea! | 04-09-2012 |

**MongoDB:**

```
{topic: 'NoSQL',
pdate: 01-09-2012,
comments: {{name: 'Peter',
           content: 'Excellent',
           cdate: 02-09-2012},
           {name: 'Tom',
           content: 'I do not understand.',
           cdate: 03-09-2012},
           {name: 'Peter',
           content: 'This is a good idea!',
           cdate: 04-09-2012}}
}
```

- keine Beziehungen zwischen Dokumenten (-> keine Joins) sondern geschachtelte Komponenten (ähnlich NF2, jedoch ohne Schemazwang)
  - Redundanz bei n:m-Beziehungen

# WAS IST MONGODB?

Hochschule RheinMain

- 2009 erstmals vorgestellt
- Skalierbare DB
- Mongo kommt von „humongous“, also riesig
- Kernziele: Performance und einfachem Datenzugriff
- Dokumenten-Datenbank, bei der Daten verschachtelt festgehalten und ad hoc abgefragt werden können
- Schema ist nicht notwendig

# HU(MONGO)US
„riesige

Hochschule RheinMain

- Gut geeignet um große Datenmengen zu speichern und gleichzeitig Ad-hoc-Queries zu verarbeiten
- Mongo ist eine JSON-Dokumenten-Datenbank (technisch als JSON-Binärformat BSON gespeichert)
- Dokumente befinden sich eigentlich im JSON-Format, das in seinem ursprünglichen Format auf ASCII-basiert und damit auch für Menschen lesbar ist, die Dokumente zur effizienten Speicherung aber im Binär-Format serialisiert sind.
- Durch das Binär-Format können weitere Daten gespeichert werden, die in der direkten JSON-Form nicht möglich sind.

# HU(MONGO)US

Hochschule RheinMain

- JSON steht für JavaScript Object Notation und ist eine kompakte menschen und maschinenlesbare Form zur Beschreibung zusammenhängender Daten.
- Die Notation wird in der Sprache JavaScript eingesetzt, wird aber mittlerweile von allen wichtigen Programmiersprachen mit Bibliotheken unterstützt.
- JSON hat gegenüber XML den Vorteil etwas kompakter zu sein, es werden weniger Bytes zur Repräsentation der gleichen Information benötigt

```
{ "name": "Tony Stark",
  "alter": 42,
  "firma": { "name": "Stark Industries",
             "ort": "New York, N.Y"
  },
  "freunde":["Steve Rogers", "Bruce Banner"]
}
```

# HU(MONGO)US

Hochschule RheinMain

- zentrale Datenstruktur Sammlungen oder Collections von JSONDokumenten
- beliebige Dokumente in einer Sammlung
- Die Sammlungen sind damit schemafrei.
- Für eine effiziente Bearbeitung ist es natürlich sinnvoll, Dokumente einer ähnlichen Struktur zu nutzen, die die gleichen Attributsnamen und für die zugehörigen Werte vergleichbare Typen haben.
- Ein Programm, das die MongoDB nutzt, kann mit mehreren Collections arbeiten.

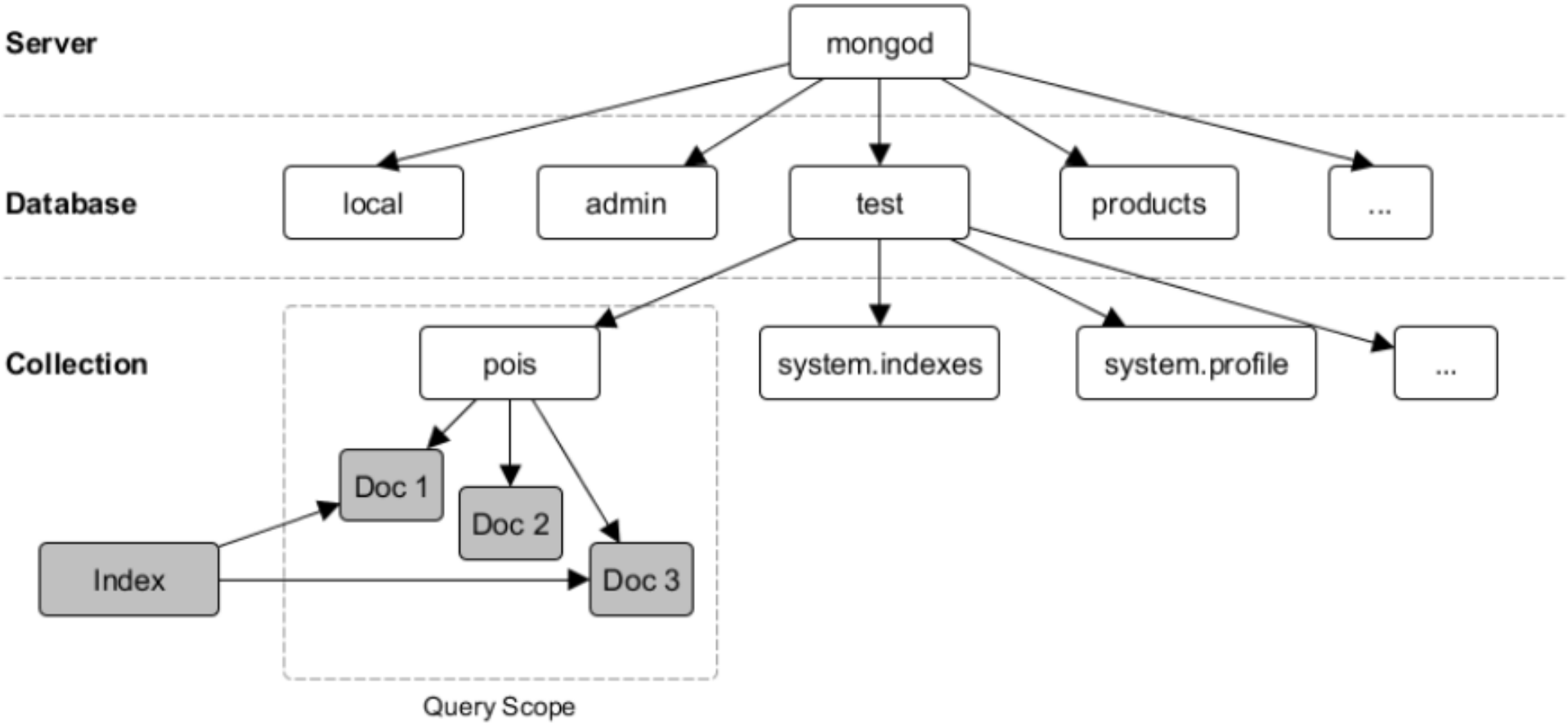
# HU(MONGO)US

Hochschule RheinMain

- MongoDB fügt jedem neu hinzugefügten Dokument automatisch ein neues Attribut _id hinzu, das als Wert einen String aus 24 Hexadezimal-Zeichen enthält, der durch seine Zusammensetzung u. a. aus der aktuellen Zeit und einem Identifikator des Computers auch über Rechnergrenzen hinweg eindeutig sein soll.
- Möchte man diesen Ansatz nicht nutzen, muss jedes hinzugefügte Dokument vor dem Hinzufügen ein Attribut _id enthalten. Der Typ des zugehörigen Wertes ist dabei egal. Der Nutzer ist dann dafür verantwortlich, dass eindeutige _id-Werte vergeben werden.
- Ist dies nicht der Fall, wird das Dokument nicht hinzugefügt und eine Fehlermeldung ausgegeben.
- Zur effizienten Bearbeitung der Dokumente wird auf dem Attribut _id automatisch ein Index erstellt, der den schnellen Zugriff sichert. Der Nutzer hat die Möglichkeit, für einzelne Attribute oder Attributkombinationen weitere Indexe zu erstellen.

# ARCHITEKTUR

Hochschule RheinMain



# CRUD

Hochschule RheinMain

Create

Read

Update

Delete

# CREATE

Hochschule RheinMain

- Neue Datenbank namens book anlegen

```
>mongo book
```

```
help
      db.help()                    help on db methods
      db.mycoll.help()             help on collection methods
      sh.help()                    sharding helpers
      rs.help()                    replica set helpers
      help admin                   administrative help
      help connect                 connecting to a db help
      help keys                    key shortcuts
      help misc                    misc things to know
      help mr                      mapreduce

      show dbs                     show database names
      show collections             show collections in current database
      show users                   show users in current database
      show profile                 show most recent system.profile entries with time >= 1ms
      show logs                    show the accessible logger names
      show log [name]              prints out the last segment of log in memory, 'global' is default
      use <db_name>                set current database
      db.foo.find()                list objects in collection foo
      db.foo.find( { a : 1 } )     list objects in foo where a == 1
      it                           result of the last line evaluated; use to further iterate
      DBQuery.shellBatchSize = x   set default number of items to display on shell
      exit                         quit the mongo shell
```

# DATENBANKEN ANZEIGEN

Hochschule RheinMain

- Show dbs
- use ???

# EINE COLLECTION ANLEGEN

## Insert()

Hochschule RheinMain

```
> db.towns.insert({
...  name: "Wiesbaden",
...  population: 320000,
...  last_census: ISODate("2012-04-15"),
...  famous_for: ["HSRM", "Oldtown"],
...  mayor: {
...  name: "Gert-Uwe Mende",
...  party:CDU}
...  }
...  )
```

# COLLECTIONEN ANZEIGEN

## Show collections

Hochschule **RheinMain**

```
> show collections
towns
```

# INHALTE EINER COLLECTION AUSGEBEN

## Find()

Hochschule **RheinMain**

```
> db.towns.find()
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population" : 320000, "last_census" : ISODate(
12-04-15T00:00:00Z"), "famous_for" : [ "HSRM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" }
```

# JOINS UND JAVASCRIPT

Hochschule RheinMain

- Mongo unterstützt keine serverseitigen Joins
- Ein einzelner JavaScript-Aufruf liefert ein Dokument und den gesamten verschachtelten Inhalt zurück
- Aber die MongoDB Muttersprache ist JavaScript. Das heißt Sie können mit MongoDB Scripte schreiben und sofort nutzen

```
function insertCity(
  name, population, last_census,
  famous_for, mayor_info
) {
  db.towns.insert({
    name:name,
    population:population,
    last_census: ISODate(last_census),
    famous_for:famous_for,
    mayor : mayor_info
  });
}
```

# AUFRUF VON CODE

Hochschule RheinMain

```
> insertCity("Punxsutaweney", 6200, '2008-31-01', ["phil the groundhog"], {name:"Jim Wehrle"}
... )
> insertCity("Portland", 58200, '2007-20-09', ["beer","food"], {name:"Sam Adams", party:"D"} )
```

# FIND

Hochschule RheinMain

- Aufruf von allen Dokumenten

**db.towns.find()**

```
> db.towns.find()
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population" : 320000, "last_census" : ISODate("20
12-04-15T00:00:00Z"), "famous_for" : [ "HSRM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" } }
{ "_id" : ObjectId("5eaee777a7bd57972b089492"), "name" : "Punxsutaweney", "population" : 6200, "last_census" : ISODate("
2010-07-01T00:00:00Z"), "famous_for" : [ "phil the groundhog" ], "mayor" : { "name" : "Jim Wehrle" } }
{ "_id" : ObjectId("5eaee7c1a7bd57972b089493"), "name" : "Portland", "population" : 58200, "last_census" : ISODate("2008
-08-09T00:00:00Z"), "famous_for" : [ "beer", "food" ], "mayor" : { "name" : "Sam Adams", "party" : "D" } }
>
```

- Aufruf eines bestimmten Dokuments

**db.towns.find({„_id“:ObjectID(„“)})**

```
> db.towns.find({"_id":ObjectId("5eaddc168a83c12961fd1dc9")})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population" : 320000, "last_census" : ISODate("20
12-04-15T00:00:00Z"), "famous_for" : [ "HSRM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" } }
>
```

# FIND

Hochschule RheinMain

• Aufruf eines bestimmten Dokuments mein einem zweitem optionalen Parameter:
ein fields-Objekt, mit dessen Hilfe sich die empfangenen Felder filtern lassen. Beispiel nur den Namen der Stadt und die ID, dann übergeben Sie name mit dem Wert 1 (oder true)
**db.towns.find({„_id“:ObjectID(„“)}, {name : 1})**

```
> db.towns.find({"_id":ObjectId("5eaddc168a83c12961fd1dc9")},{name:1})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden" }
>
```

Um alle Felder außer name abzurufen, setzen Sie name auf 0 oder false oder null
**db.towns.find({„_id“:ObjectID(„“)}, {name : 0})**

```
>
> db.towns.find({"_id":ObjectId("5eaddc168a83c12961fd1dc9")},{name:0})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "population" : 320000, "last_census" : ISODate("2012-04-15T00:00:00Z"),
"famous_for" : [ "HSRM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" } }
>
```

# FIND 2
## Beispiel

Hochschule RheinMain

- Alle Städte mit dem Anfangsbuchstaben P und weniger als 10000 Einwohner

```
> db.towns.find(
... {
... name:/^P/, population: {$lt:10000}
... }
... )
{ "_id" : ObjectId("5eaee777a7bd57972b089492"), "name" : "Punxsutaweney", "population" : 6200, "last_census" : ISODate("
2010-07-01T00:00:00Z"), "famous_for" : [ "phil the groundhog" ], "mayor" : { "name" : "Jim Wehrle" } }
>
```

- Alle Städte mit dem Anfangsbuchstaben P und Einwohner weniger als 400000 und mehr als 10000 – Aufbau als ob es Objekte wären

```
> var populations_range = {}
> populations_range['$lt'] = 400000
400000
> populations_range['$gt'] = 10000
10000
```

```
> db.towns.find(
... {
... name: /^P/,
... population:populations_range},
... {name:1}
... )
{ "_id" : ObjectId("5eaee7c1a7bd57972b089493"), "name" : "Portland" }
```

# FIND 2
## Kombinationen und Bereiche integrieren

Hochschule **RheinMain**

- Bedingsoperatoren haben das Format
feld : {$op : wert}

| Name | Description |
|---|---|
| `$eq` | Matches values that are equal to a specified value. |
| `$gt` | Matches values that are greater than a specified value. |
| `$gte` | Matches values that are greater than or equal to a specified value. |
| `$in` | Matches any of the values specified in an array. |
| `$lt` | Matches values that are less than a specified value. |
| `$lte` | Matches values that are less than or equal to a specified value. |
| `$ne` | Matches all values that are not equal to a specified value. |
| `$nin` | Matches none of the values specified in an array. |

# FIND 2
## Datenbereich

Hochschule RheinMain

```
> db.towns.find(
...     {
...     last_census :
...     {$lte : ISODate('2008-08-09')}
...     },
...     {
...     _id:0, name:1}
...     )
{ "name" : "Portland" }
>
```

# FIND 2
## Partielle Treffer

Hochschule RheinMain

```
> db.towns.find( { famous_for:/groundhog/}, {_id:0, name:1, famous_for:1})
{ "name" : "Punxsutaweney", "famous_for" : [ "phil the groundhog" ] }
>
```

# FIND 2

## Alle passenden Werte und keine passenden Werte

Hochschule **RheinMain**

```
> db.towns.find( { famous_for: {$all:['food', 'beer']}}, {_id:0, name:1, famous_for:1})
{ "name" : "Portland", "famous_for" : [ "beer", "food" ] }
```

```
> db.towns.find( { famous_for: {$nin:['food', 'beer']}}, {_id:0, name:1, famous_for:1})
{ "name" : "Wiesbaden", "famous_for" : [ "HSRM", "Oldtown" ] }
{ "name" : "Punxsutaweney", "famous_for" : [ "phil the groundhog" ] }
>
```

# FIND 3

## Ergebnisse von verschachtelten Subdokumenten

Hochschule RheinMain

- Feldnamen als String eingeben und die verschachtelten Ebenen durch Punkte voneinander zu trenne

- Städte mit Bürgermeister der CDU

```
> db.towns.find( {'mayor.party':'CDU'})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population" : 320000, "last_census" : ISODate("20
12-04-15T00:00:00Z"), "famous_for" : [ "HSRM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" } }
```

- Städte mit nicht bekannten Parteien der Bürgermeister

```
> db.towns.find( {'mayor.party':{$exists:false}}, {_id:0, name:1, mayor:1})
{ "name" : "Punxsutaweney", "mayor" : { "name" : "Jim Wehrle" } }
{ "name" : "New York", "mayor" : { "name" : "Bill de Blasio" } }
>
```

Hochschule RheinMain

# ÜBUNGEN

Die ersten Schritte mit MongoDB

# ÜBUNG 1/1

Hochschule RheinMain

1. Laden Sie sich einen MongoDB-Server herunter und installieren Sie ihn. (https://www.mongodb.com/download-center/community)
2. Erstellen Sie eine Datenbank und wiederholen Sie die Schritte die Ihnen gezeigt wurden.
3. Fügen Sie 5 weitere Städte hinzu – Sie dürfen sich gerne alle Daten ausdenken
4. Erstellen Sie eine Query die Ihnen alle Städte zeigt, die bekannt sind für die HSRM.
5. Erstellen Sie eine Query die Ihnen alle Städte zeigt, die bekannt sind NUR für die HSRM.
6. Erstellen Sie eine Query die Ihnen alle Städte zeigt, wo es keinen bekannten Bürgermeister gibt
7. Erstellen Sie eine Query die Ihnen alle Städte zeigt, wo der Bürgermeister ein er enthält.

Hochschule RheinMain

# SHOW YOUR WORK

# ELEM MATCH
## Länder speichern

Hochschule **RheinMain**

- Hinzufügen von Ländern

```
> db.countries.insert({ _id:"us", name:"United States", exports: { food :[ {name:
"bacon", tasty:true},{name:"burgers"}]}})
WriteResult({ "nInserted" : 1 })
```

```
> db.countries.insert({ _id:"ca", name:"Canada", exports: { food :[ {name:"bacon"
, tasty:false},{name:"syrup", tasty:true}]}})
WriteResult({ "nInserted" : 1 })
>
```

```
> db.countries.insert({ _id:"de", name:"Germany", exports: { food :[ {name:"bacon
", tasty:true},{name:"bread", tasty:true}]}})
WriteResult({ "nInserted" : 1 })
>
```

# ANZAHL VON EINTRÄGEN

## Count()

Hochschule **RheinMain**

```
> print(db.countries.count())
3
>
```

# FINDEN

## Suche nach Bacon der lecker ist

Hochschule **RheinMain**

- 1. Versuch – leider fehlerhaft

```
> db.countries.find( {'exports.food.name':'bacon', 'exports.food.tasty':true},  {
_id:0, name:1} )
{ "name" : "United States" }
{ "name" : "Canada" }
{ "name" : "Germany" }
>
```

2. Versuch mit $elem-Match, es legt fest, dass ein Dokument als Treffer zählt, wenn all unsere Kriterien erfüllt sind

```
> db.countries.find( {'exports.food':{$elemMatch:{name:'bacon', 'tasty':true}}},
 {_id:0, name:1} )
{ "name" : "United States" }
{ "name" : "Germany" }
>
```

# BOOLESCHE OPERATOREN

Und

Oder

Hochschule RheinMain

- UND

```
>
> db.countries.find({_id:"de", name:"United States"}, {_id:true})
>
```

- ODER

```
> db.countries.find({$or:[{_id:"de"}, {name:"United States"}]}, {_id:true})
{ "_id" : "us" }
{ "_id" : "de" }
>
```

# BEFEHLE

Hochschule **RheinMain**

## Logical

| Name | Description |
| --- | --- |
| `$and` | Joins query clauses with a logical **AND** returns all documents that match the conditions of both clauses. |
| `$not` | Inverts the effect of a query expression and returns documents that do *not* match the query expression. |
| `$nor` | Joins query clauses with a logical **NOR** returns all documents that fail to match both clauses. |
| `$or` | Joins query clauses with a logical **OR** returns all documents that match the conditions of either clause. |

# BEFEHLE

Hochschule RheinMain

### Element

| Name | Description |
| --- | --- |
| `$exists` | Matches documents that have the specified field. |
| `$type` | Selects documents if a field is of the specified type. |

### Evaluation

| Name | Description |
| --- | --- |
| `$expr` | Allows use of aggregation expressions within the query language. |
| `$jsonSchema` | Validate documents against the given JSON Schema. |
| `$mod` | Performs a modulo operation on the value of a field and selects documents with a specified result. |
| `$regex` | Selects documents where values match a specified regular expression. |
| `$text` | Performs text search. |
| `$where` | Matches documents that satisfy a JavaScript expression. |

# BEFEHLE
## Und noch mehr

Hochschule RheinMain

- https://docs.mongodb.com/manual/reference/operator/query/

Hochschule RheinMain

# ÜBUNGEN

Erweiterte Read-Operationen mit MongoDB

# ÜBUNG 1/1

Hochschule RheinMain

1. Erstellen Sie insgesamt 5 Länder + die 3 hier vorgestellten
2. Finden Sie das Land welches Bier exportiert
3. Finden Sie das Land welches Bier und Bacon exportiert
4. Welches Land enthält ein e und hat keine Exporte die NICHT tasty:false sind

Hochschule RheinMain

# SHOW YOUR WORK

# UPDATE

## Update(kriterium, operation)

Hochschule **RheinMain**

- Hinzufügen von Feld Bundesstaat in unseren Städten

```
> db.towns.update( {_id:ObjectId("5eaddc168a83c12961fd1dc9")}, {$set:{"state":"HE
"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.towns.find( {_id:ObjectId("5eaddc168a83c12961fd1dc9")})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population
" : 320000, "last_census" : ISODate("2012-04-15T00:00:00Z"), "famous_for" : [ "HS
RM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" }, "stat
e" : "HE" }
>
```

- Einwohner um 1000 erhöhen

```
>
> db.towns.update( {_id:ObjectId("5eaddc168a83c12961fd1dc9")}, {$inc:{population:
1000}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.towns.find( {_id:ObjectId("5eaddc168a83c12961fd1dc9")})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population
" : 321000, "last_census" : ISODate("2012-04-15T00:00:00Z"), "famous_for" : [ "HS
RM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" }, "stat
e" : "HE" }
>
```

# UPDATE BEFEHLE

https://docs.mongodb.com/manual/reference/operator/update/

Hochschule RheinMain

### Fields

| Name | Description |
|---|---|
| `$currentDate` | Sets the value of a field to current date, either as a Date or a Timestamp. |
| `$inc` | Increments the value of the field by the specified amount. |
| `$min` | Only updates the field if the specified value is less than the existing field value. |
| `$max` | Only updates the field if the specified value is greater than the existing field value. |
| `$mul` | Multiplies the value of the field by the specified amount. |
| `$rename` | Renames a field. |
| `$set` | Sets the value of a field in a document. |
| `$setOnInsert` | Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents. |
| `$unset` | Removes the specified field from a document. |

# UPDATE BEFEHLE

Hochschule **RheinMain**

### Operators

| Name | Description |
|---|---|
| `$` | Acts as a placeholder to update the first element that matches the query condition. |
| `$[]` | Acts as a placeholder to update all elements in an array for the documents that match the query condition. |
| `$[<identifier>]` | Acts as a placeholder to update all elements that match the `arrayFilters` condition for the documents that match the query condition. |
| `$addToSet` | Adds elements to an array only if they do not already exist in the set. |
| `$pop` | Removes the first or last item of an array. |
| `$pull` | Removes all array elements that match a specified query. |
| `$push` | Adds an item to an array. |
| `$pullAll` | Removes all matching values from an array. |

# UPDATE BEFEHLE

Hochschule **RheinMain**

### Modifiers

| Name | Description |
|---|---|
| `$each` | Modifies the `$push` and `$addToSet` operators to append multiple items for array updates. |
| `$position` | Modifies the `$push` operator to specify the position in the array to add elements. |
| `$slice` | Modifies the `$push` operator to limit the size of updated arrays. |
| `$sort` | Modifies the `$push` operator to reorder documents stored in an array. |

### Bitwise

| Name | Description |
|---|---|
| `$bit` | Performs bitwise **AND**, **OR**, and **XOR** updates of integer values. |

# REFERENZEN

{$ref:“collection_name“, $id:“reference_id“}

Hochschule **RheinMain**

- Städte den Ländern zuordnen

```
> db.towns.update( {_id:ObjectId("5eaddc168a83c12961fd1dc9")}, {$set:{country:
... {$ref:"countries", $id:"de"}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

```
> db.towns.find( {_id:ObjectId("5eaddc168a83c12961fd1dc9")})
{ "_id" : ObjectId("5eaddc168a83c12961fd1dc9"), "name" : "Wiesbaden", "population
" : 321000, "last_census" : ISODate("2012-04-15T00:00:00Z"), "famous_for" : [ "HS
RM", "Oldtown" ], "mayor" : { "name" : "Gert-Uwe Mende", "party" : "CDU" }, "stat
e" : "HE", "country" : DBRef("countries", "de") }
>
```

# REFERENZEN NUTZEN

Hochschule RheinMain

```
> var wiesbaden = db.towns.findOne({_id:ObjectId("5eaddc168a83c12961fd1dc9")})
> db.countries.findOne({_id:wiesbaden.country.$id})
{
        "_id" : "de",
        "name" : "Germany",
        "exports" : {
                "food" : [
                        {
                                "name" : "bacon",
                                "tasty" : true
                        },
                        {
                                "name" : "bread",
                                "tasty" : true
                        }
                ]
        }
}
>
```

# LÖSCHEN
## remove

Hochschule RheinMain

- Ersetzen Sie die find-Funktion durch Remove
- Es wird das ganze Dokument entfernt
- Einmal ausgeführt, wird nicht mehr nachgefragt

```
> var bad_bacon = { 'exports.food': { $elemMatch: { name:'bacon',  tasty:false }
} }
>
>
> db.countries.find(bad_bacon)
{ "_id" : "ca", "name" : "Canada", "exports" : { "food" : [ { "name" : "bacon", "
tasty" : false }, { "name" : "syrup", "tasty" : true } ] } }
>
```

```
>
> db.countries.remove(bad_bacon)
WriteResult({ "nRemoved" : 1 })
>
```

```
> db.countries.find()
{ "_id" : "us", "name" : "United States", "exports" : { "food" : [ { "name" : "ba
con", "tasty" : true }, { "name" : "burgers" } ] } }
{ "_id" : "de", "name" : "Germany", "exports" : { "food" : [ { "name" : "bacon",
"tasty" : true }, { "name" : "bread", "tasty" : true } ] } }
>
```

Hochschule RheinMain

# ÜBUNGEN

Update und Delete

# ÜBUNG 1/1

Hochschule RheinMain

1. Fügen Sie New York zu ihren Städten hinzu.

2. Wählen Sie eine Stadt über einen die Groß- und Kleinschreibung ignorierenden regulären Ausdruck, der das Wort new enthält. Löschen Sie diese Stadt.

3. Erweiteren Sie Wiesbaden um das Feld “bekannte Studenten” und fügen Sie sich ein.

4. Erstellen Sie eine Collection Student mit Name, Studiengang und HSRM-Kürzel. Füllen Sie diese Tabelle mit 6 Studenten. Fügen Sie diese Referenz als neues Feld (alle Studenten) in Wiesbaden ein.

Hochschule RheinMain

# SHOW YOUR WORK

# DIE ABSCHLUSSABGABE

Hochschule RheinMain

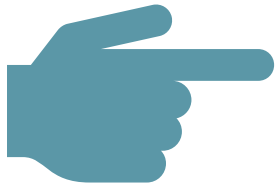
- Aus subato unter Ressources finden Sie „The little MongoDB-Book“ [1] und „MongoDB Professional“ in der Aufgabe

1. Arbeiten Sie sich durch Kapitel 1 von [1] durch und erstellen Sie die Collection Unicorn mit den dort vorgegebenen Daten. Erstellen Sie Queries für folgende Fragen:
   1. Welche Unicorns (name und gender) sind vor 1980 geboren?
   2. Wieviele Unicorns lieben Äpfel (appel)?
   3. Wieviele Unicorns lieben NUR Äpfel (appel)?
   4. Was ist das durchschnittliche Gewicht aller weiblichen unicorns?
   5. Welche Unicorns haben ein o im Namen?
   6. Geben Sie die Namen von allen Weibchen, welche Äpfel oder Karotte mögen, aus. Sortiert nach Gewicht.
   7. Welche Unicorns haben bis jetzt noch keine Vampire getötet, sortiert nach Geschlecht.
   8. Welche Männchen haben ein Gewicht zwischen 600 und 900?
2. Arbeiten Sie sich durch Kapitel 2 von [1].
   1. Es ist Corona-Zeit. Erhöhen Sie das Gewicht aller Unicorns um 10.
   2. Alle männlichen lieben jetzt zusätzlich Chocolade und die Anzahl der getöteten Vampire reduziert sich um 10

# MAP REDUCE
## In Kurzform

Hochschule RheinMain

Aufgabe definieren



Mapper erstellen

Reducer definieren

# BERICHT ERSTELLEN

Der alle Telefonnummern zählt, die nach Land die gleichen Ziffern enthalten

Hochschule RheinMain

- 1. Punkt – Hilfsfunktion, die ein Array aller unterschiedlicher Nummern erzeugt

```
distinctDigits = function(phone){
  var
    number = phone.components.number + '',
    seen = [],
    result = [],
    i = number.length;
  while(i--) {
    seen[+number[i]] = 1;
  }
  for (i=0; i<10; i++) {
    if (seen[i]) {
      result[result.length] = i;
    }
  }
  return result;
}
db.system.js.save({_id: 'distinctDigits', value: distinctDigits})
```

# MAPPER UND REDUCER

Hochschule RheinMain

```
map = function() {
  var digits = distinctDigits(this);
  emit({digits : digits, country : this.components.country}, {count : 1});
}
```

```
reduce = function(key, values) {
  var total = 0;
  for(var i=0; i<values.length; i++) {
    total += values[i].count;
  }
  return { count : total };
}
```

# AUFRUF

Hochschule RheinMain

```
results = db.runCommand({
mapReduce:'phones',
map: map,
reduce: reduce,
out: 'phones.report'
})

db.phones.report.find({'_id.country':8})
```

# REDUCER 2

## Wenn Ausgabe total statt count

Hochschule RheinMain

```
reduce = function(key, values) {
  var total = 0;
  for(var i=0; i<values.length; i++) {
    var data = values[i];
    if('total' in data) {
      total += data.total;
    } else {
      total += data.count;
    }
  }
  return { total : total };
}
```

# QUELLENVERZEICHNIS

Hochschule RheinMain


1. Sieben Wochen, sieben Datenbanken von Eric Redmond, Jim R. Wilson
2. Professional NoSQL von Shashank Tiwari
3. Little MongoDB Book
4. MongoDB Notes for Professionals
5. https://docs.mongodb.com/