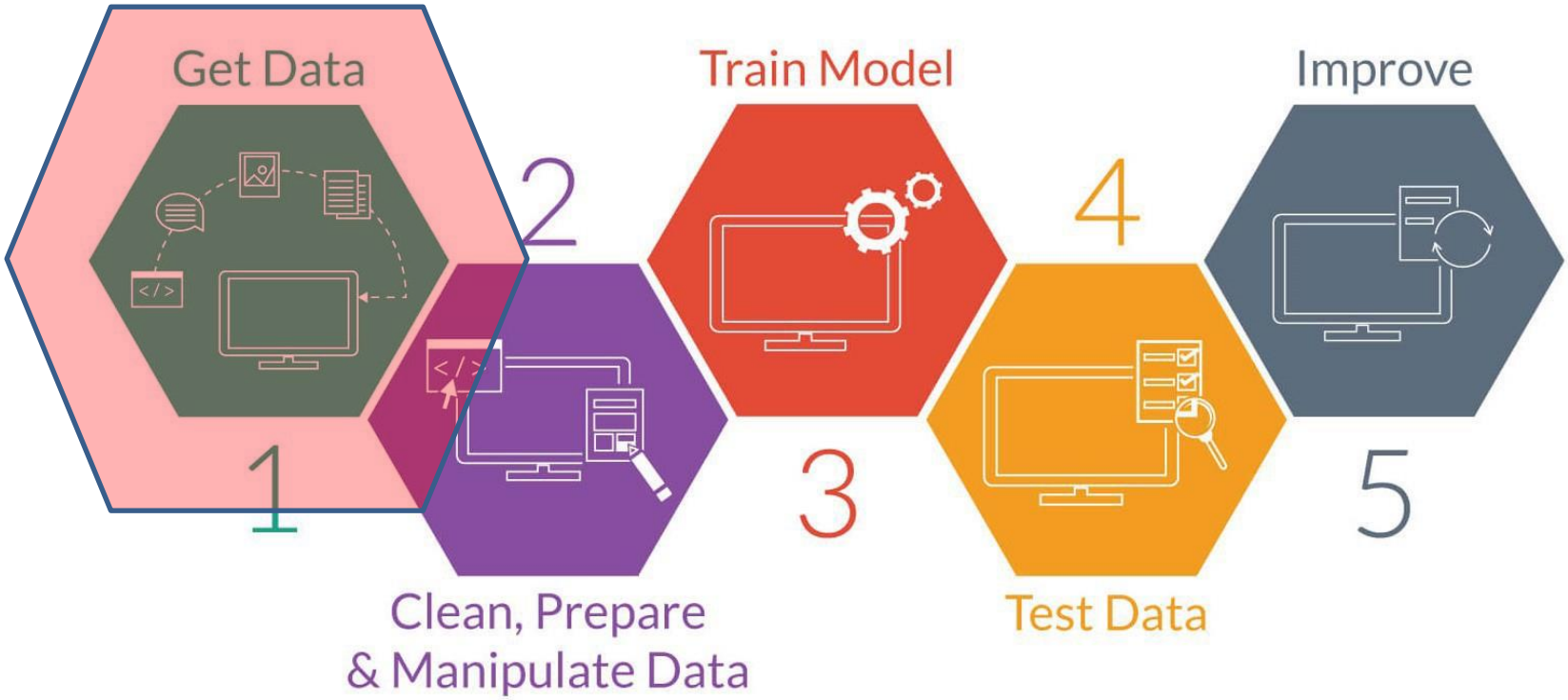
Lecture 4: MongoDB Overview

Big Data System Design

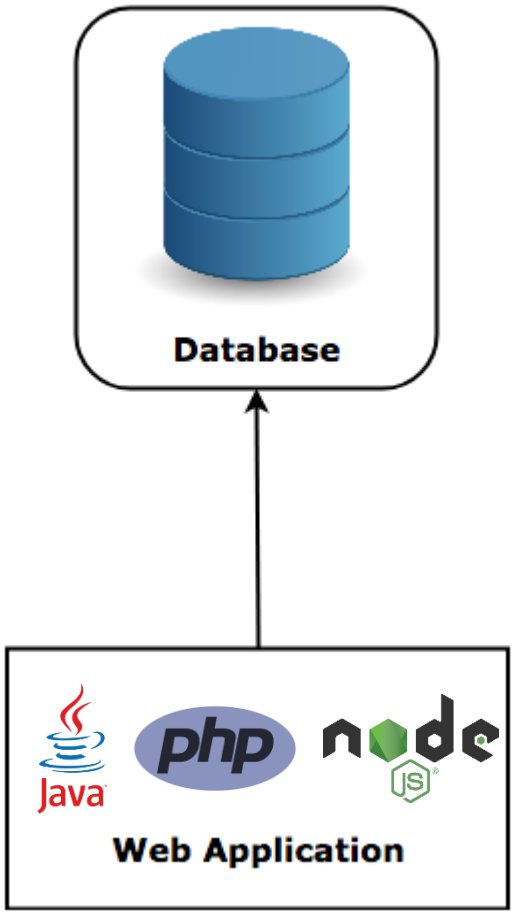
In Last Lecture

- Big data process



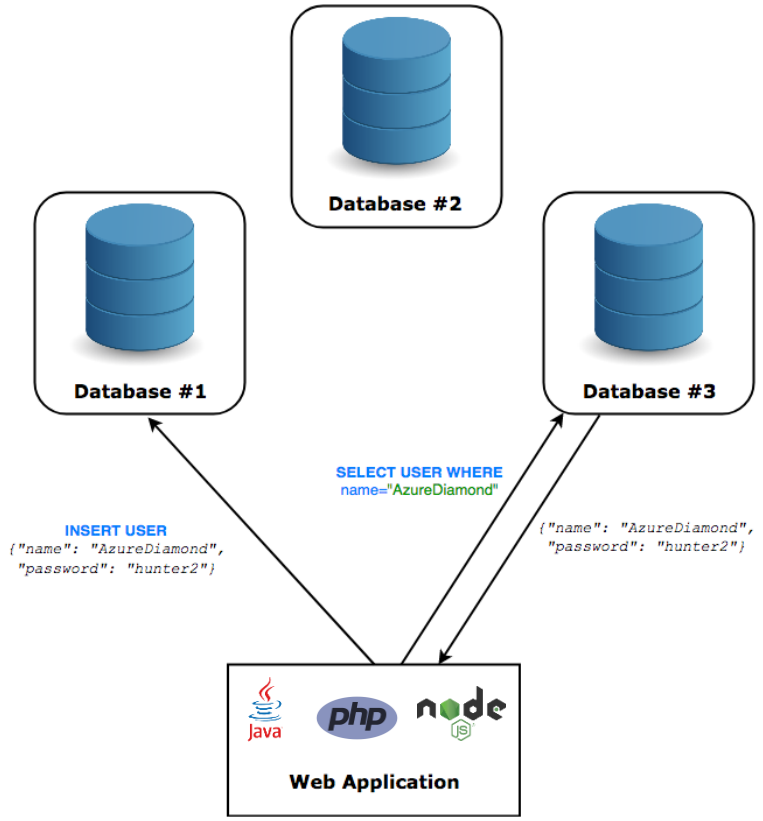
In Last Lecture

- Centralized storage
  - Data is stored on the database of one single machine
  - Whenever you want to read/insert information in it you communicate with that machine directly
  - Relational databases
    - Schema



In Last Lecture

- Decentralized storage
  - Database run on multiple machines at the same time
  - User does not know if he is talking to single machine or multiple machines
  - NoSQL databases
    - Schemeless



In Last Lecture

- Why NoSQL?
  - Scalability
  - Fault-tolerance
  - Schema-less
  - Large companies are switching to NoSQL
  - Open source

In Last Lecture

- Famous NoSQL Databases

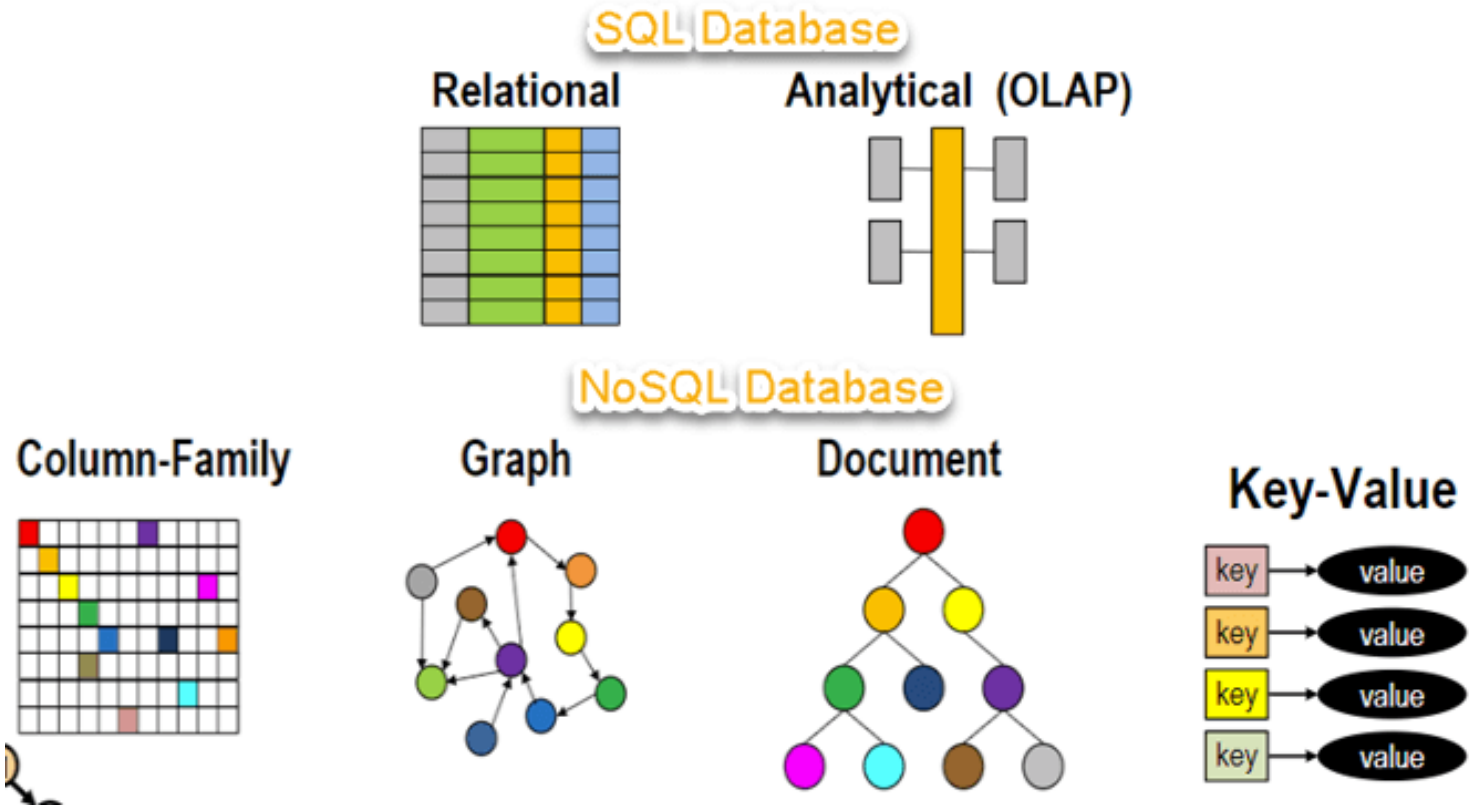


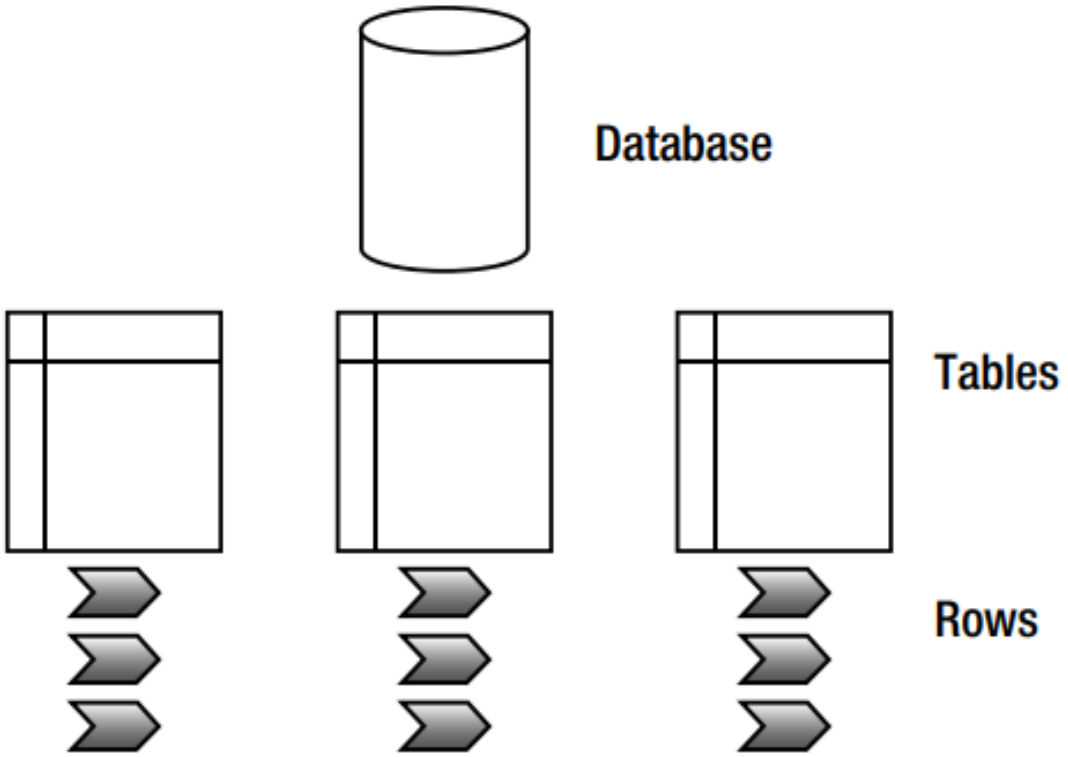
Table of Contents

Part 1

MONGODB OVERVIEW

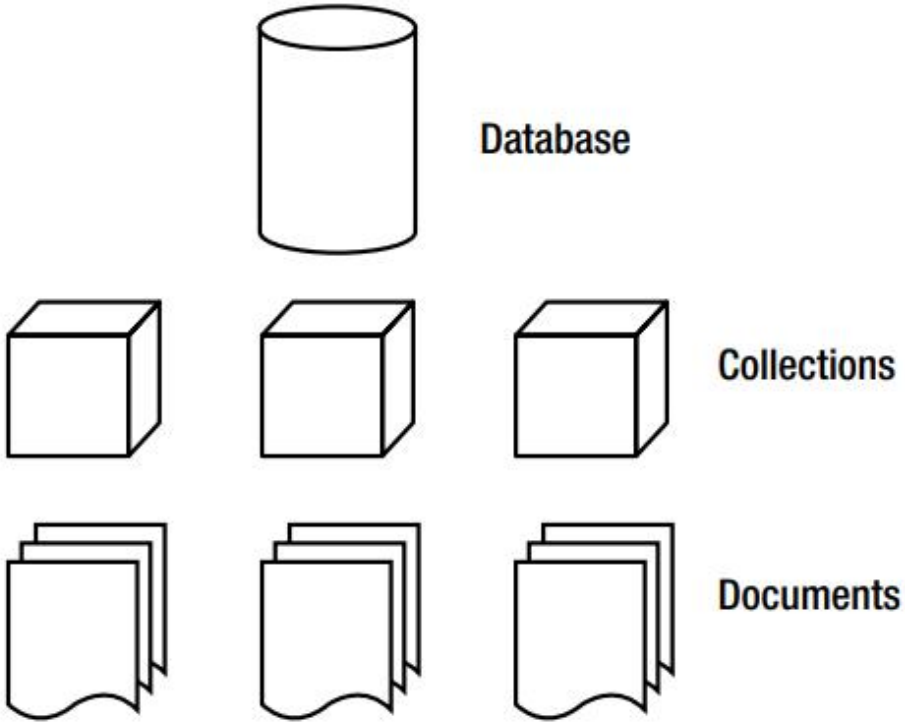
MongoDB Overview

- A typical relational database model
  - One table holds different rows



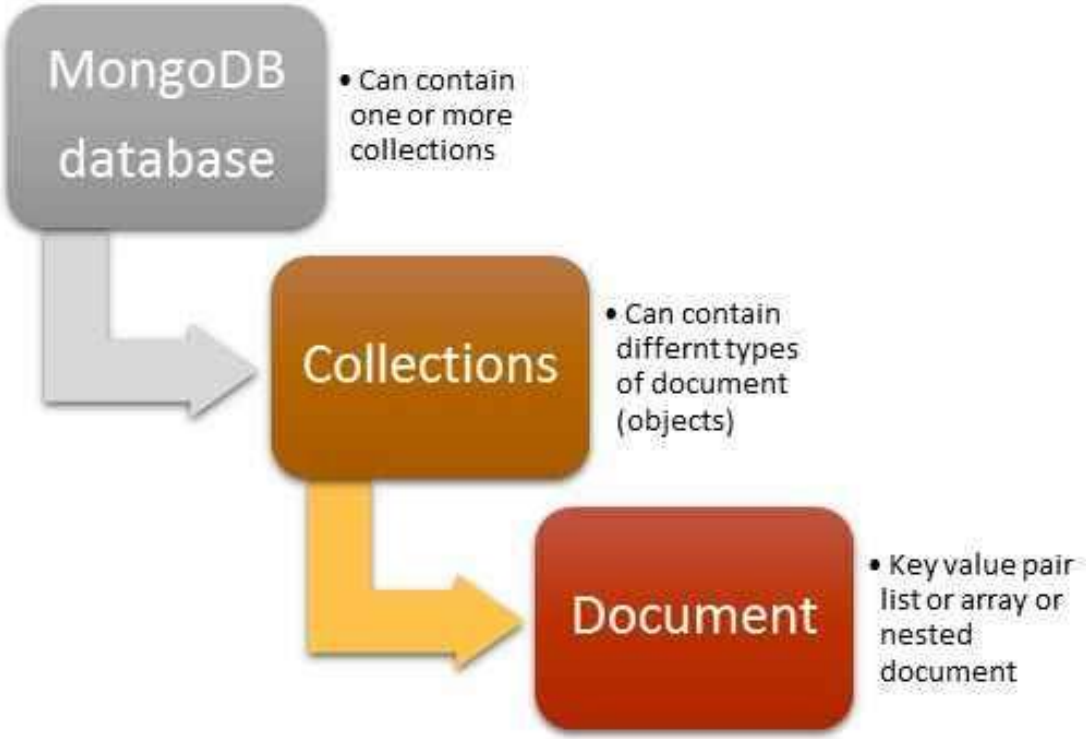
MongoDB Overview

- MongoDB
  - A document database in which one collection holds different documents



MongoDB Overview

- MongoDB components



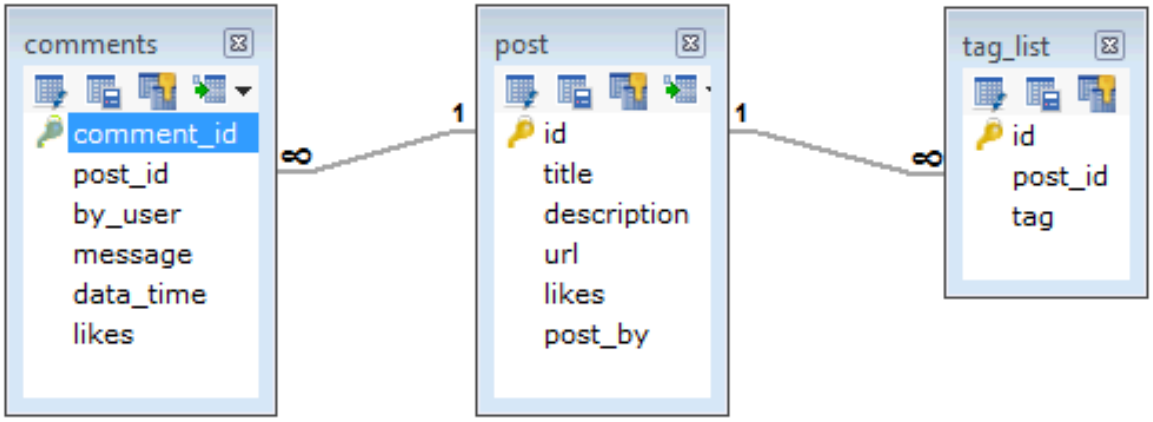
MongoDB Overview

- Relationship of RDBMS terminology with MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

MongoDB Overview

- Suitable for big data
  - Unstructured data
    - Suppose a client needs a database design for his blog/website
      - In RDBMS schema, design for the requirements will have minimum three tables



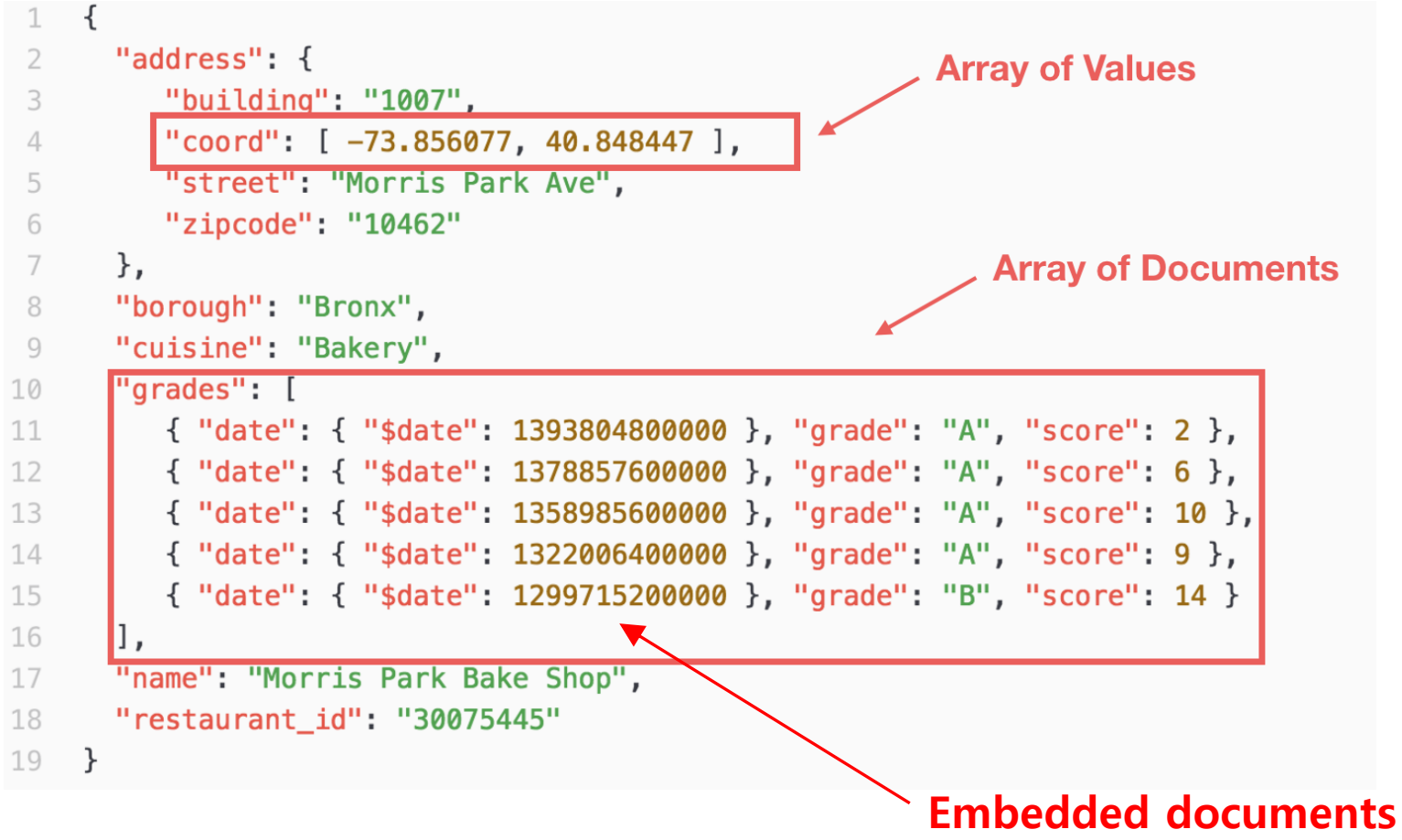
MongoDB Overview

- Example
  - In MongoDB schema, design will have one collection post and the following structure (Aggregate orientation)

```
{
   _id: POST_ID
   title: TITLE_OF_POST,
   description: POST_DESCRIPTION,
   by: POST_BY,
   url: URL_OF_POST,
   tags: [TAG1, TAG2, TAG3],
   likes: TOTAL_LIKES,
   comments: [
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      },
      {
         user:'COMMENT_BY',
         message: TEXT,
         dateCreated: DATE_TIME,
         like: LIKES
      }
   ]
}
```

MongoDB Overview

- Embedded document



MongoDB Overview

- MongoDB document pattern
  - 1:1 pattern

Students

Student_id	Student_name
2023299001	Jeong-Hun Kim
2023299002	Jae-Seong Yeon
2023299003	Jong-Pil Park

Personal information

Registration number	Phone	Student_id
990101-X	01012341234	2023299001
990615-X	01015152020	2023299002
990930-X	01019194848	2023299003

```
{
    "_id": "2023299001",
    "name": "Jeong-Hun Kim",
    "Personal information":
    {
        "Registration number": "990101-X",
        "Phone": "01012341234"
    }
}
```

Embedded documents

MongoDB Overview

❖ MongoDB document pattern

- 1:N pattern

Students

Student_id	Student_name
2023299001	Jeong-Hun Kim
2023299002	Jae-Seong Yeon
2023299003	Jong-Pil Park

Grade

Lecture	Score	Student_id
Database	A+	2023299001
Database	B+	2023299002
Data structure	A+	2023299001

① Embedded document (strong association)

```
{
    "_id": "2023299001",
    "name": "Jeong-Hun Kim",
    "Personal information":
    {
        "Registration number": "990101-X",
        "Phone": "01012341234"
    },
    "Grades":[
        {"lecture": "Database", "score": "A+"},
        {"lecture": "Data structure", "score": "A+"}
    ]
}
```

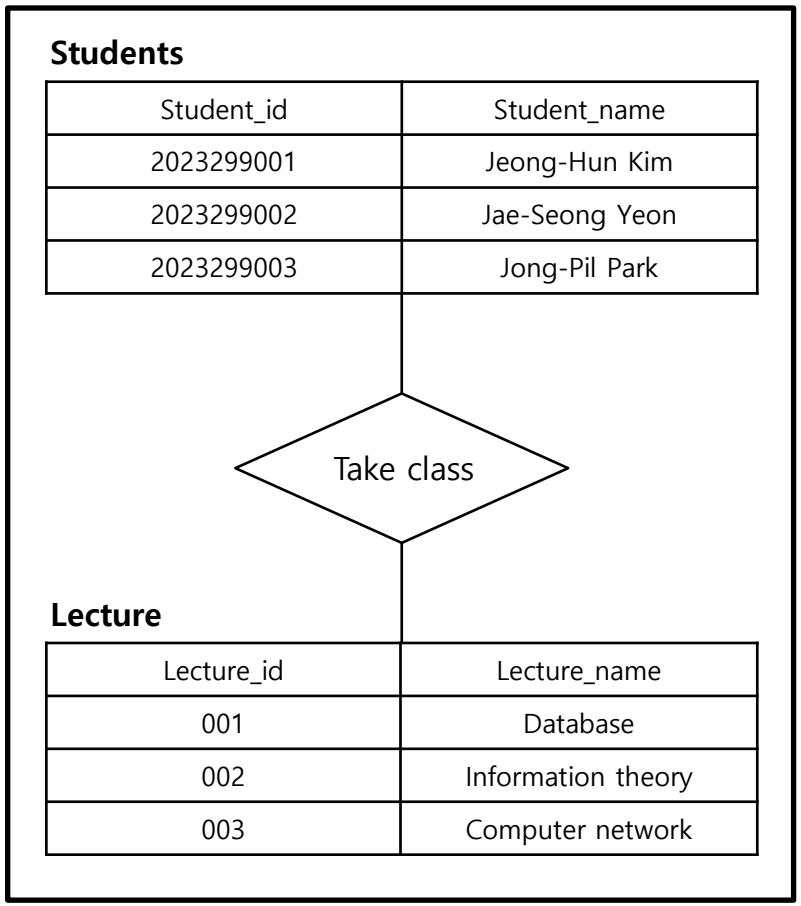
② Linked document (weak association)

```
{
    "_id": "2023299001",
    "name": "Jeong-Hun Kim",
    "Personal information":
    {
        "Registration number": "990101-X",
        "Phone": "01012341234"
    }
}
```

```
{"_id": "001", "lecture": "Database", "score": "A+",
"student_id": "2023299001"}
{"_id": "002", "lecture": "Data structure", "score": "A+",
"student_id": "2023299001"}
```

MongoDB Overview

- MongoDB document pattern
  - N:M pattern



MongoDB Overview

- MongoDB document pattern
  - N:M pattern

    ① One-way

```
[Students]
{"_id": "2023299001", "name": "Jeong-Hun Kim", "class": ["001", "002", "003", ...]}
{"_id": "2023299002", "name": "Jae-Seong Yeon", "class": ["001", 003", ...]}
{"_id": "2023299003", "name": "Jong-Pil Park", "class": ["002", ...]}

[Lecture]
{"_id": "001", "name": "Database"}
{"_id": "002", "name": "Information theory"}
{"_id": "003", "name": "Computer network"}
```

MongoDB Overview

- MongoDB document pattern
  - N:M pattern

    ② Two-way

```
[Students]
{"_id": "2023299001", "name": "Jeong-Hun Kim", "class": ["001", "002", "003", ...]}
{"_id": "2023299002", "name": "Jae-Seong Yeon", "class": ["001", 003", ...]}
{"_id": "2023299003", "name": "Jong-Pil Park", "class": ["002", ...]}

[Lecture]
{"_id": "001", "name": "Database", "students": ["2023299001", "2023299002", ...]}
{"_id": "002", "name": "Information theory", "students": ["2023299001", "2023299003"]}
{"_id": "003", "name": "Computer network", "students": ["2023299001", "2023299002"]}
```

MongoDB Overview

- MongoDB document pattern
  - Tree pattern

```
a
|-b
   |-c
   |-d
|- e
   |- f
```

Access speed ↑

Complexity ↑

```
{
 _id : tree,
 name : "a",
 childs : [
   {
     name : "b",
     childs ": [
       { name : "c" },
       { name : "d" }
       ]
     }, {
     name : "e",
     childs : [
       { name : "f" }
       ]
   }
 ]
}
```

Reference: https://blog.voidmainvoid.net/241

MongoDB Overview

- MongoDB document pattern
  - Tree pattern

```
a
|-b
   |-c
   |-d
|- e
   |- f
```

```
{ _id: "a" }
{ _id: "b", ancestors: [ "a" ], parent: "a" }
{ _id: "c", ancestors: [ "a", "b" ], parent: "b" }
{ _id: "d", ancestors: [ "a", "b" ], parent: "b" }
{ _id: "e", ancestors: [ "a" ], parent: "a" }
{ _id: "f", ancestors: [ "a", "e" ], parent: "e" }
```

Reference: https://blog.voidmainvoid.net/241

Difficulty of updating the tree ↑

MongoDB Overview

- MongoDB document pattern
  - Dynamic field pattern
    - Store data in field names

```
{
  _id: "S001",
  name : "홍길동",
  courses: [
    { coursename : "국어", score : 90, instructor:"김샘" },
    { coursename : "수학", score : 80, instructor:"박샘" },
    ...
  ]
}
```

```
{
  _id: "S001",
  name : "홍길동",
  courses: {
    "국어" : { score: 90, instructor:"김샘" },
    "수학" : { score: 80, instructor:"박샘" },
    ...
  },
  clist : ["국어", "수학" ]
}
```

Reference: https://blog.voidmainvoid.net/241

MongoDB Overview

❖ Embedded document vs. linked document

Embedded document	Property	Linked document
Strong	Relationship	Weak
Not required	(Strict) consistency	Required
Small	Document size	Big
Less	Update frequency	Many

Part 2

MONGODB COMPONENTS

Database

- What is database?
  - Database is a physical container for collections
  - Each database gets its own set of files on the file system
- Create database
  - MongoDB uses the following command to create a database
    - *use DATABASE_NAME*
      - If the database already exists, it will return the existing database

Database

- Create database
  - If you want to create a database with name 'mydb', then 'use DATABASE_NAME' statement would be as follows:
    - **use mydb**

    switched to db mydb

  - To check your currently selected database, use the command 'db'
    - **db**

    mydb

  - If you want to check your databases list, use the 'show dbs'
    - **show dbs**

    local 0.78125GB

    test 0.23012GB

Database

- Create database
  - Your created database (mydb) is not present in list
    - Because, it is empty for now
  - To display database, you need to insert at least one document into it (*note, it can be any document)
    - **db.mycollection.insert({name: "Jeong-Hun Kim"})**
    - **show dbs**

    local 0.78125GB

    test 0.23012GB

    mydb 0.23012GB

Database

- Drop database
  - MongoDB 'db.dropDatabase()' command is used to drop an existing database
    - **db.dropDatabase()**
  - If you want to delete another database, then you should first switch to that database and then do 'db.dropDatabase()'
    - **use socialnetwork**

```
switched to db socialnetwork
```

    - **db.dropDatabase()**

```
{ "dropped" : "socialnetwork", "ok" : 1 }
```

Collection

- Create collection
  - Collection is a group of MongoDB documents similar to tables of RDBMS
  - Syntax
    - *db.createCollection(name, options)*
      - Name is name of collection to be created
      - Options is a document and is used to specify configuration of collection

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Collection

- Create collection
  - Basic syntax of 'createCollection()' method without options is as follows
    - **use mydb**

switched to db mydb

    - **db.createCollection("newcollection")**

{ "ok" : 1 }

  - You can check the created collection by using the command 'show collections'
    - **show collections**

mycollection

newcollection

Collection

- Create collection
  - MongoDB creates collection automatically, when you insert some document
    - **db.anothercollection.insert({name : "New Document"})**

WriteResult({ "nInserted" : 1 })

- **show collections**

newcollection

mycollection

anothercollection

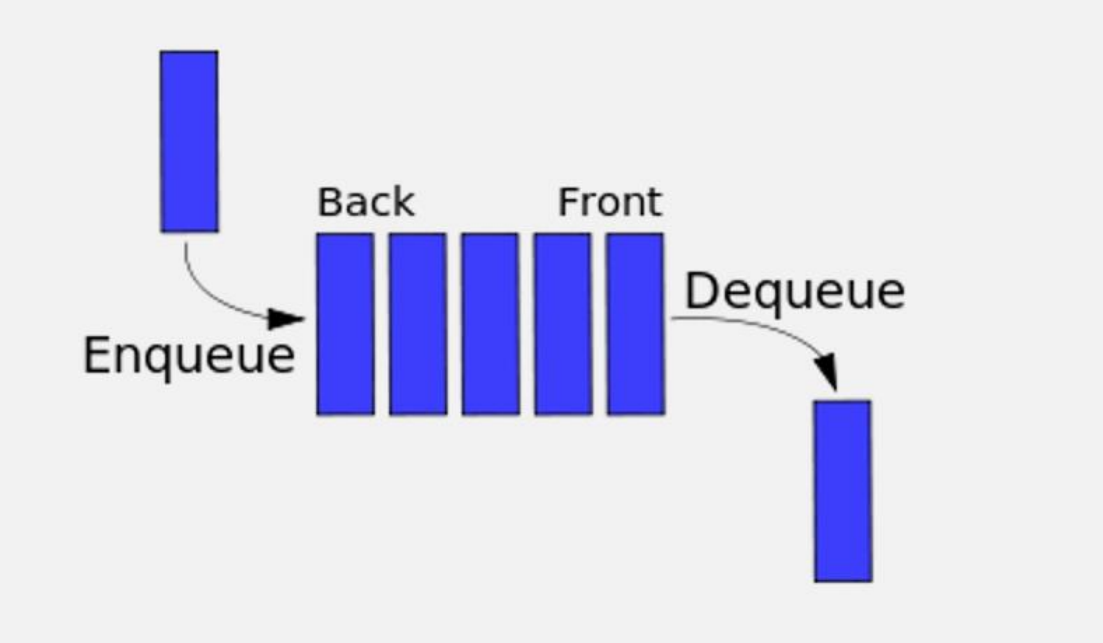
Collection

- Capped collection
  - *db.createCollection(name, options)*
    - Options parameter is optional
      - You need to specify only the name of the collection

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

Collection

- Capped collection
  - To create a capped collection
    - **db.createCollection("cappedLogCollection",{capped:true, size:10000})**



Collection

- Capped collection
  - The 'max' for limiting the number of documents in the collection
    - **db.createCollection("cappedLogCollection",{capped:true, size:10000,max:1000})**
  - If you want to check whether a collection is capped or not
    - **db.cappedLogCollection.isCapped()**
  - If you want to convert an existing collection to the capped
    - **db.runCommand({"convertToCapped":"collName", size:10000})**
  - If you want to change options of a capped collection
    - **db.runCommand({"collMod":"collName", cappedSize: 100000})**
    - **db.runCommand({"collMod":"collName", cappedMax: 500})**

Collection

- Collation
  - A set of rules that define how strings are compared and sorted
    - The character set of MongoDB is UTF-8

```
> db.char_test.find({},{_id:0}).sort({col:1})
{ "col" : "A1" }
{ "col" : "B1" }
{ "col" : "C1" }
{ "col" : "a1" }
{ "col" : "b1" }
{ "col" : "Á1" }
{ "col" : "á1" }
{ "col" : "Ć1" }
{ "col" : "ć1" }
```

Reference: https://hoing.io/archives/4639

Collection

- Collation
  - Options

Option	Description
locale	Regional/language information optimized for specific countries and languages.
caseLevel	Capitalization and accent
caseFirst	Priority for uppercase and lowercase letters
Strength	Strenth for comparing strings
numericOrdering	Whether the numeric string is numeric or character
Alternate	Spaces and punctuation
maxVariable	Strings to exclude from comparison
backwards	Direction to compare
normalization	Whether text normalization is required

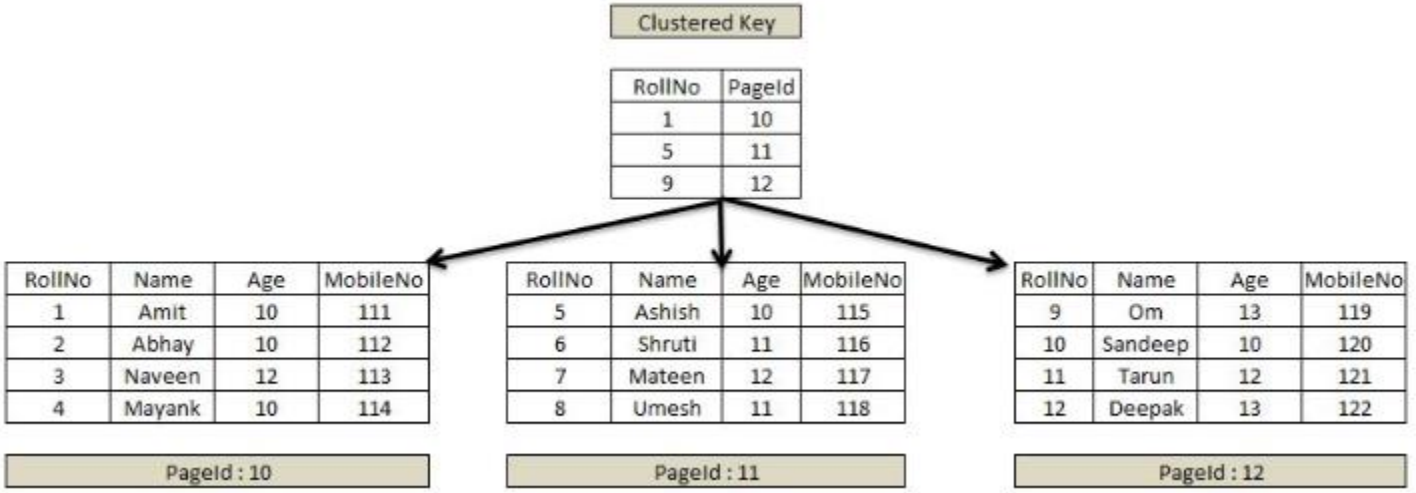
Collection

- Time series collection
  - Time series data is a sequence of data in which insights are gained by analyzing changes over time
  - Time series collections efficiently store time series data

  - Benefits
    - Reduced complexity for working with time series data
    - Improved query efficiency
    - Reduced disk usage
    - Reduced I/O for read operations
    - Increased WiredTiger cache usage

Collection

- Clustered collection
  - Collections created with a clustered index
  - Clustered index: Physical rearrangement



Reference: https://velog.io/@gillog/SQL-Clustered-Index-Non-Clustered-Index

Collection

- Clustered collection
  - Benefits
    - Faster queries on clustered collection without needing a secondary index
    - Lower storage size (Improving performance for queries and bulk inserts)
    - No secondary TTL (Time To Live) index required
    - Additional performance improvements for inserts, updates, deletes, and queries

Collection

- How to drop collection
  - MongoDB's 'db.collection.drop()' is used to drop a collection from the database
  - Basic syntax of 'drop()' command is as follows:
    - *db.COLLECTION_NAME.drop()*
  - First check the available collections and drop the collection with the name *anothercollection*
    - **db.anothercollection.drop()**

true

Document

- Insert a document
  - To insert one or more data objects into MongoDB collection, you need to use MongoDB's 'insert()' method
  - The basic syntax of 'insert()' command is as follows
    - *db.COLLECTION_NAME.insert(document)*
      - Document is {key: value} or {field: value}
  - Example
    - **db.artists.insert({ artistname: "Jorn Lande" })**

```
WriteResult({ "nInserted" : 1 })
```

Document

- Insert multiple documents
  - **db. artists.insert(**
    [
    { artistname: "The Kooks" },
    { artistname: "Bastille" },
    { artistname: "Gang of Four" }
    ]
)

```
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        …
})
```

Document

- insertOne()
  - You can also use the insertOne() method to insert a single document into a collection
  - The basic syntax of 'insert()' command is as follows
    - *db.COLLECTION_NAME.insertOne(document)*
  - Example
    - **db.musicians.insertOne({ _id: 1, name: "Ian Gillan", instrument: "Vocals" })**

{ "acknowledged" : true, "insertedId" : 1 }

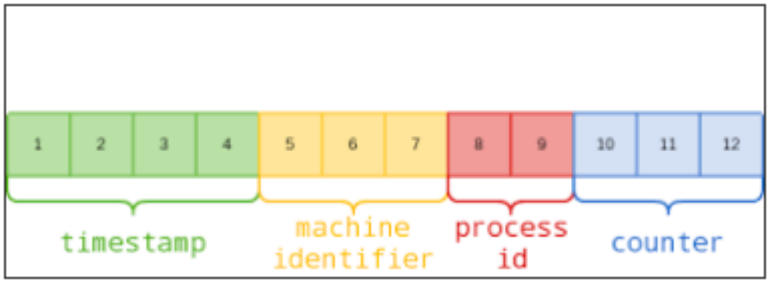
Document

- insertMany()
  - As the name suggests, you can use insertMany() to insert multiple documents
  - The basic syntax of 'insert()' command is as follows
    - *db.COLLECTION_NAME.insertMany(documents)*
  - Example
    - **db.musicians.insertMany(**

```
[
  { _id: 2, name: "Ian Paice", instrument: "Drums", born: 1948 },
  { _id: 3, name: "Roger Glover", instrument: "Bass", born: 1945 },
  { _id: 4, name: "Steve Morse", instrument: "Guitar", born: 1954 },
]
)
```

Document

- "_id" field
  - Role of unique key
  - If not explicit, ObjectID is automatically provided



Reference: https://blog.voidmainvoid.net/239

Query Document

- Select all documents in a collection
  - To query the musicians collection, simply define the following select criteria
    - **db. musicians.find()**
  - To display the results in a formatted way, you can use pretty() method
    - *db.COLLECTION_NAME.find().pretty()*
    - **db. musicians.find().pretty()**
      - Compare difference between them
  - This operation corresponds to the following SQL statement
    - SELECT * FROM musicians

Summary and Discussions

- MongoDB Overview
  - Database, collection, document, field, embedded documents

- Database
  - Commands
    - use DATABASE_NAME
    - show dbs
    - db.dropDatabase()

  - Don’t forget that you must insert at least a document for database to be displayed

Summary and Discussions

- Collection
  - Collection is a group of MongoDB documents similar to tables of RDBMS
  - Commands
    - db.createCollection(name, options)
    - show collections
    - db.COLLECTION_NAME.drop()
  - Capped collections
    - A fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size

Summary and Discussions

- Document
  - A document is a set of key-value pairs
  - MongoDB provides the following three methods for inserting documents into a database:
    - insert()
    - insertOne()
    - insertMany()
  - Commands
    - db.COLLECTION_NAME.insert(document)

Questions?

SEE YOU NEXT TIME!