# Big Data (MHI222956/MHI225101)

## 3.2 MongoDB – a document-oriented database

# MongoDB

- A document-oriented database, also known as *document store database*, *aggregate database*, or *simply document store* or *document database*
- A database that uses a document-oriented model to store data
- Document–oriented databases store each record and its associated data within a single document.
- Each document contains semi-structured data that can be queried against using various query and analytics tools
- MongoDB stores data records as BSON documents. BSON is a binary representation of JSON documents.

# JSON

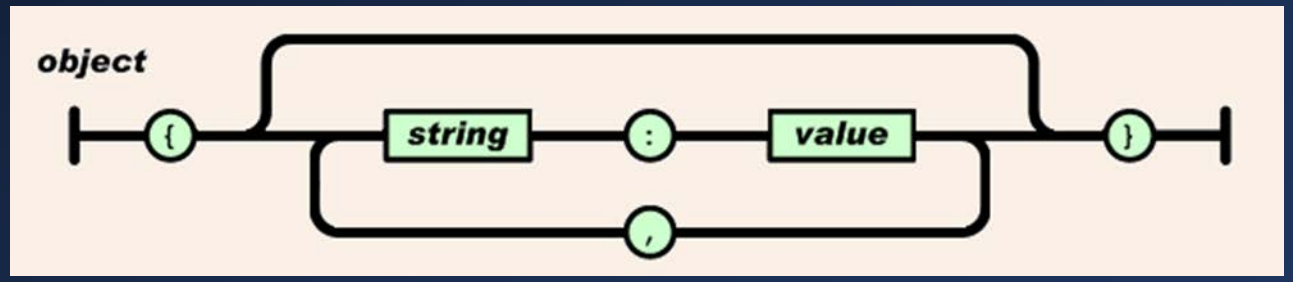
- JSON (JavaScript Object Notation, http://www.json.org/) is a lightweight data-interchange format.
- JSON is text, written with JavaScript object notation.

- JSON is built on two structures:
  - A collection of name/value pairs.
    - In various languages, this is realized as an object, record, structure, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values.
    - In most languages, this is realized as an array, vector, list, or sequence.

- Virtually all modern programming languages support them in one form or another.

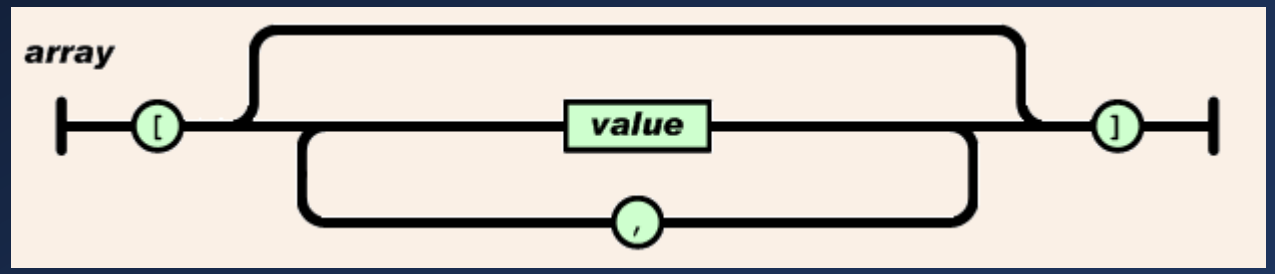
# JSON - object

- An object is an unordered set of name/value pairs.
  - An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



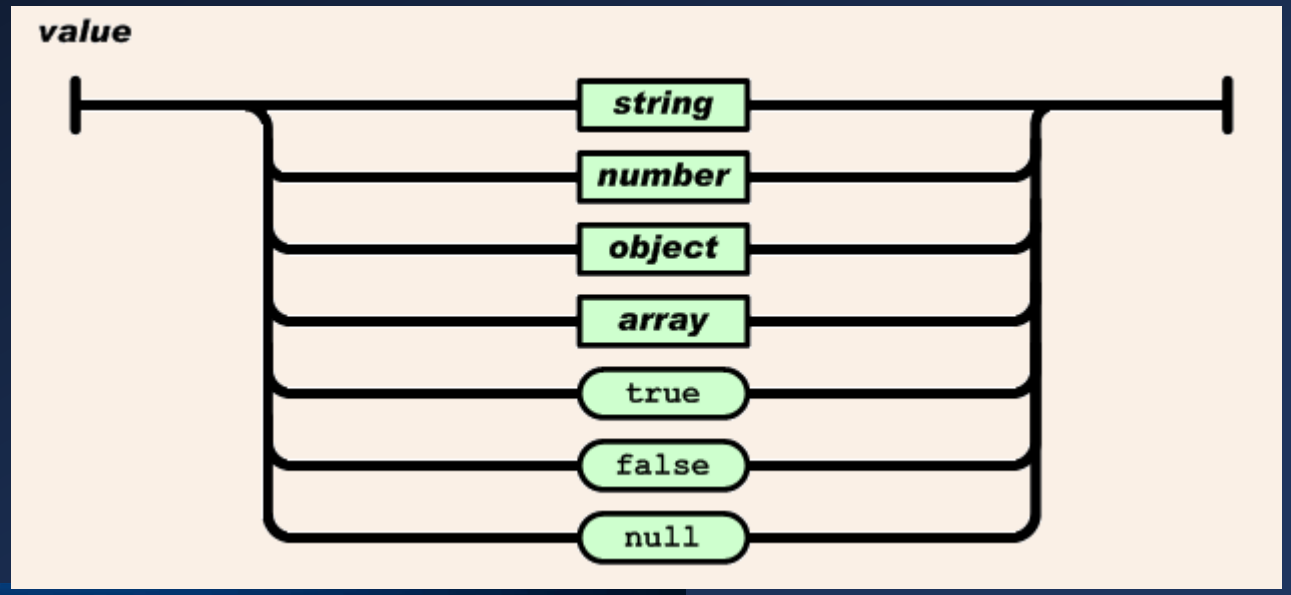
# JSON - array

- An array is an ordered collection of values.
  - An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).



# JSON - value

- A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.



# JSON example

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

- How many objects are there in this JSON example?
  - 5
  - 6 ✓
  - 8
  - 11
  - Other answers

# MongoDB Document Structure

- MongoDB documents are composed of field-and-value pairs and have the following structure:

```
{
    field1: value1,
    field2: value2,
    field3: value3,
    ...
    fieldN: valueN
}
```

```
{
    _id: ObjectId("5099803df3f4948bd2f98391"),
    name: { first: "Alan", last: "Turing" },
    birth: new Date('Jun 23, 1912'),
    death: new Date('Jun 07, 1954'),
    contribs: [ "Turing machine", "Turing test", "Turingery" ],
    views : NumberLong(1250000)
}
```

- The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents.

| Data type | Description |
|---|---|
| *String* | text values, e.g. “Glasgow” |
| *Boolean* | TRUE or FALSE. |
| *Integer (32b and 64b)*: | numerical value, e.g. 5 |
| *Double* | floating point values, e.g. 5.24 |
| *Min / Max keys* | values which will always compare lower / higher than any other type |
| *Timestamp* | used to store a timestamp, 64-bit integer |
| *Null* | used for a Null value |
| *Date* | used to store the current date or time in UNIX time format (POSIX time), 64-bit integer |
| *Object ID* | used to store the document’s ID, generated by database |
| *Binary data* | used to store binary data. |
| *Regular expression* | |
| *JavaScript Code* | |

# MongoDB Document Structure

- Field names are strings.
- Restrictions on field names:
  - The field name *_id* is reserved for use as a primary key; its value must be unique in the collection, is immutable, and may be of any type other than an array.
  - The field names **cannot** start with the dollar sign ($) character.
  - The field names **cannot** contain the dot (.) character.
  - The field names **cannot** contain the null character.
- Field Value Limit

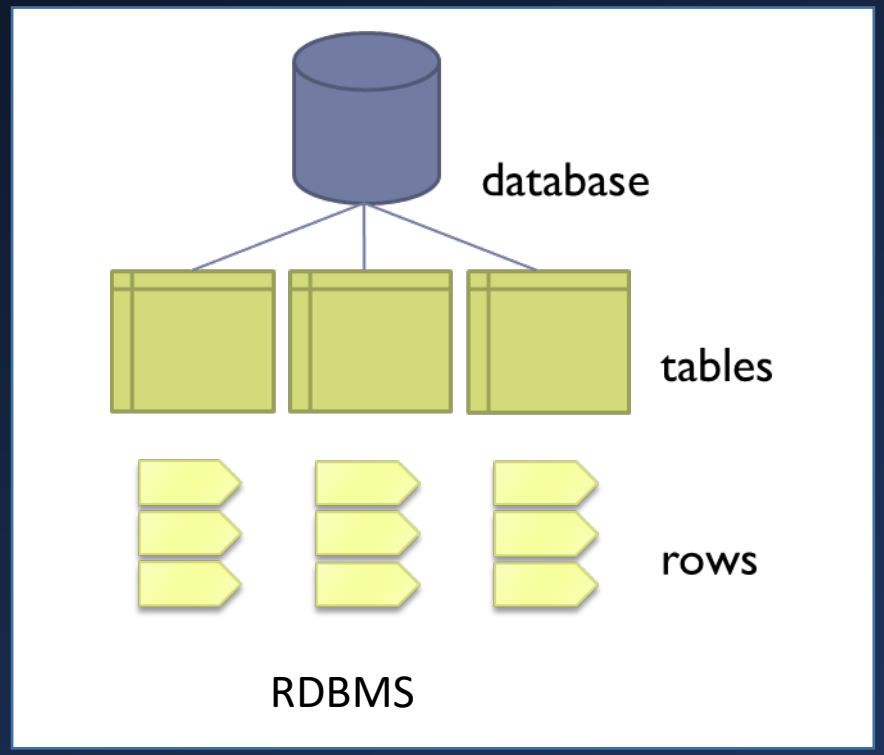
  For indexed collections, the values for the indexed fields have a Maximum Index Key Length limit.

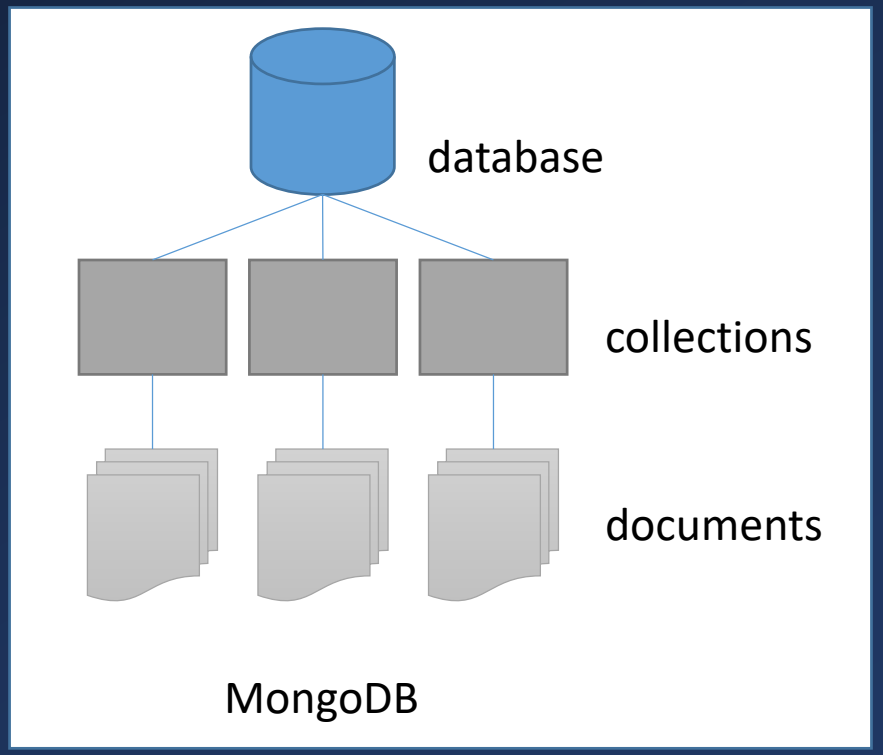
# MongoDB vs Relational Database

- Terminology

| 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 |

- Model


database
tables
rows
RDBMS


database
collections
documents
MongoDB

- RDMBS
  - Meaning of data items in each row is defined by the design of the table that contains the row
  - Column names, types, constraints
  - Database has a well defined structure, or schema, for the data it can store
  - Database will reject data that doesn't conform to schema

- MongoDB
  - Meaning of data items in a document is contained in the document itself – name part of name/value pairs
  - Database/collections have no knowledge of meaning of data
  - Database (like many NoSQL stores) is ***schemaless***
  - Database will allow documents with any structure to be stored
  - Can store unstructured data where the content is not known apriori and can vary from document to document

# Getting start with MongoDB

- Installation
  - Download from:
    https://www.mongodb.com/download-center?initial=true#community
  - Install manual:
    https://docs.mongodb.com/manual/administration/install-community/

- Tools in bin folder after installation:
  - |-- bin
  - | |-- mongo (the database shell)
  - | |-- mongod (the core database server)
  - | |-- mongos (auto-sharding process)
  - | |-- mongodump (dump/export utility)
  - | |-- mongorestore (restore/import utility)

- MongoDB requires a data folder to store its files. MongoDB will not start if a data directory doesn’t exist.

- Default data directory is:
  - /data/db (Linux, MacOS)
  - c:\data\db (Windows)

- May need to set permissions to allow user to read & write

- Start server with command at system command prompt:

  mongod --dbpath data/db

  - --dbpath option allows you to specify data directory, can omit if using the default location, example here assumes data directory is data/db in the current working directory
  - Assuming here that mongodb bin folder is in path, may need to specify full path otherwise

- You should see startup messages including something like this, tells you the server is alive and what TCP port number is being used (27017)

  2018-09-31T14:36:40.701+0000 I NETWORK [thread1] waiting for connections on port 27017

- Note that you now need to keep the terminal window open, server will shut down if you close it
- You can also install and run MongoDB as a service, detached from any terminal window

- From another terminal on the same computer, enter the following command to run mongo shell client:

```
mongo
```

  - This assumes that there is no security configured in the database, which is OK for learning. But in a “real” installation, username/password would need to specify – see documentation
  - Will look for server instance running on localhost at default port (27017)
  - Should see the following message and the > shell prompt (may see other messages also):

```
MongoDB shell version v3.4.0
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.0
>
```

- MongoDB server can also be accessed from a remote machine with mongo client installed, e.g.

```
mongo --host 192.168.231.127 --port 27017
```

- The shell client is always available if MongoDB is installed
- Mongo shell provides a way to interact with the database through commands

- Separate tools are available that provide a GUI or web interface, e.g. MongoClient, interact with the database.
  (https://docs.mongodb.com/ecosystem/tools/)

# MongoDB CRUD Operations

- Creating databases/collections using the shell

```
db                          shows the name of the current database
use myDatabase              switches to a different database, creates if it doesn't exist
db.myCollection.insert( {"country":"Scotland"} )
                            inserts a document in the specified collection in the current
                            database, creates collection if it doesn't exist
```

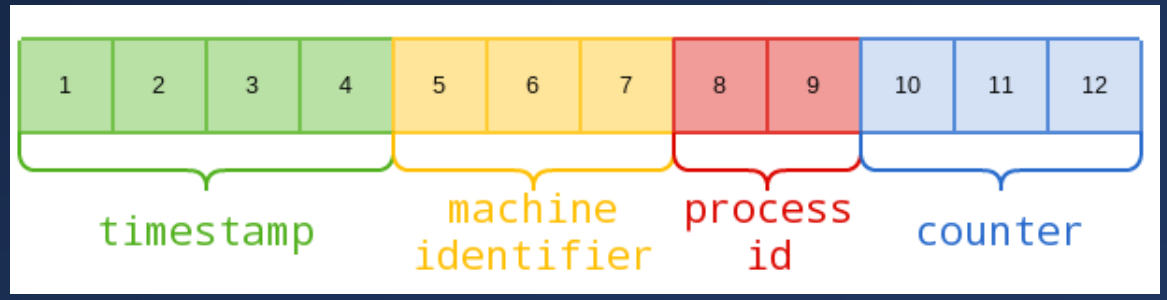
- Since version 3.2, MongoDB provides the following methods to insert documents into a collection:

```
db.myCollection.insertOne()
db.myCollection.insertMany()
```

- You can check that the document has been stored using the *find* function :

```
> db.myCollection.find()
{ "_id" : ObjectId("59e01dd19400bbe099b3b35d"), “country" : “Scotland" }
{ "_id" : ObjectId("59e01ea59400bbe099b3b35e"), “country" : “England",
"city" : “London" }
{ "_id" : ObjectId("59e0200d9400bbe099b3b35f"), "city" : “Washington",
“sport" : “basketball", “country" : “USA"}
```

- We didn’t have to specify what properties a document can have, and that different documents can have different fields.
- MongoDB adds an *_id* property to each document, if you do not specify the _id value manually, then the type will be set to a special BSON datatype that consists of a 12-byte binary value:



- Update an existing document by adding a new property:

```
db.myCollection.updateOne(
  {“country" : “USA"},
  {$set : {“city“ : “New York"}})
```

- Delete a specific document:

```
db.myCollection.remove({“city“ : “London"})
```

- Deleting a whole collection from the database:

```
db.myCollection.drop()
```

# Data Retrieval in RDBMS

- Query language
  - A language in which a retrieval request is specified, i.e., to specify the data items you need .
- SQL – Structured Query Language
  - The ubiquitous query language when the data is structured
- Basic structure of an SQL query:

  SELECT (output attribute(s))
  FROM (table(s) to use)
  WHERE (the condition(s) to satisfy)

# Data Retrieval in MongoDB

- MongoDB does not support a query language such as SQL
- Main query mechanism is the *find()* function

- The basic syntax of find( ) method is:

  >db.collection.find( )

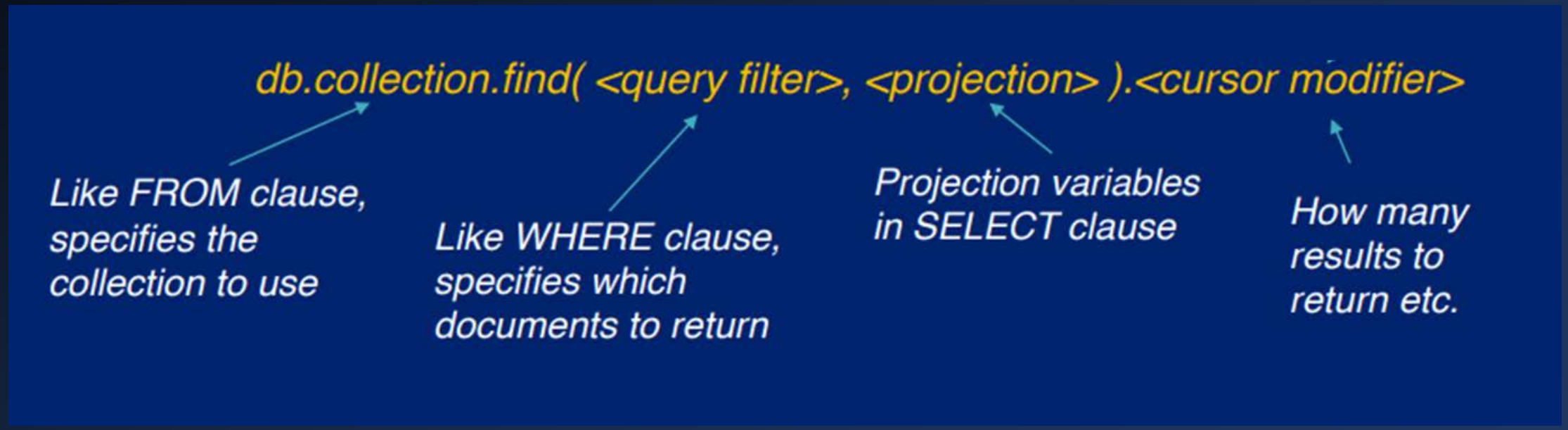
  all the documents in the collection will be displayed in a non-structure way.

- The pretty() method:

  >db.collection.find( ).pretty()

  documents will be displayed in a formatted way.

# SQL select and MongoDB find()



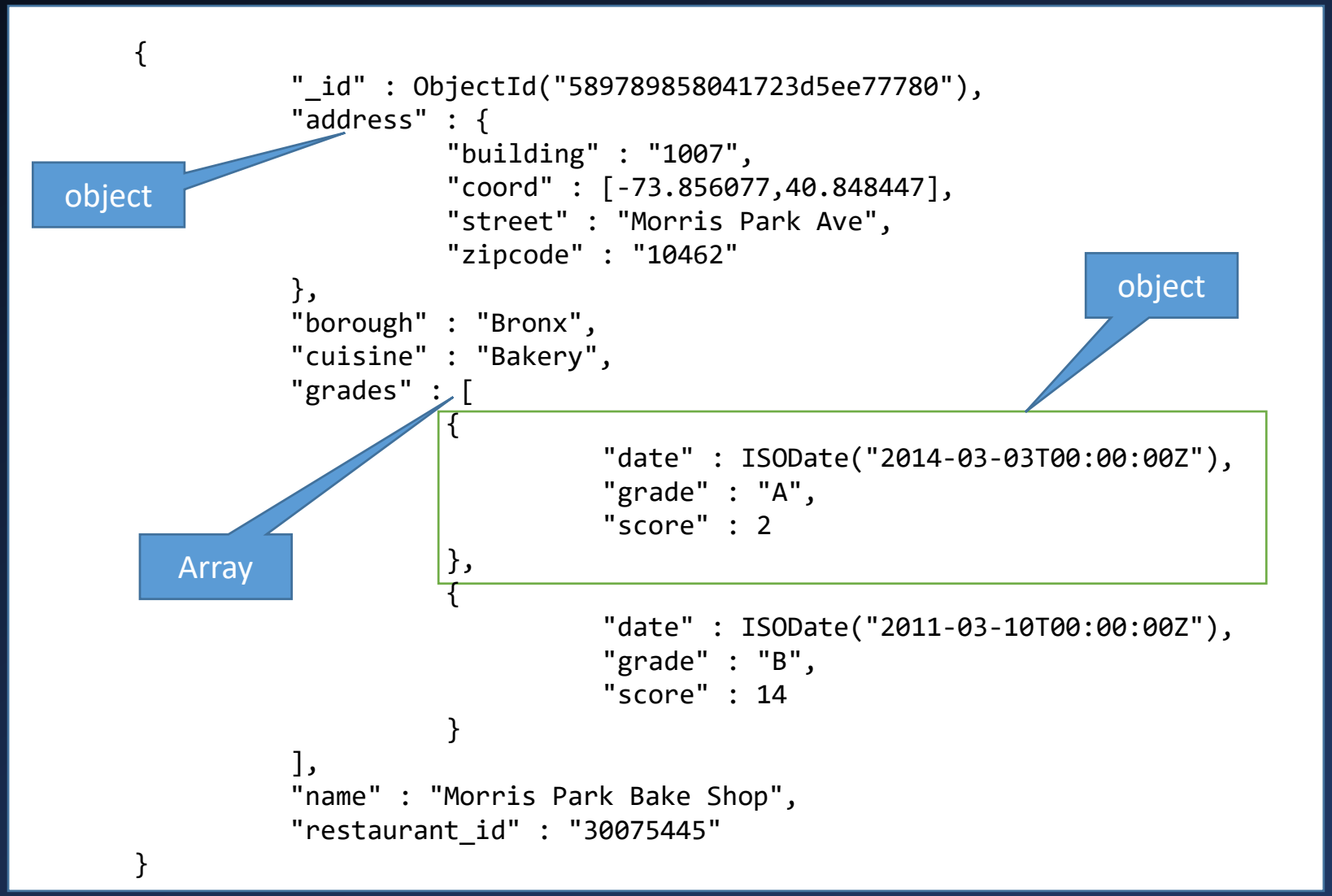

# The find() Function

- To build a query on a collection, you specify a query filter (or, query document) with properties you wish the results to match and pass this as a parameter to *find*
- Query document is a JSON document whose properties are either field names of documents in the collection or MongoDB operators - operators have names that start with $, e.g. *$lt, $exists*
- Other functions can be chained with *find* to process the results, e.g. *sort*, *limit, pretty*

- RDBMS Where Clause Equivalents in MongoDB

| Operation | Syntax | Example | RDBMS Equivalent |
|---|---|---|---|
| Equality | {<key> : <value>} | db.mycol.find( {"by":"tutorials point"}) | where by = 'tutorials point' |
| Less Than | {<key> : {$lt:<value>}} | db.mycol.find( {"likes":{$lt:50}}) | where likes < 50 |
| Less Than Equals | {<key> : {$lte:<value>}} | db.mycol.find( {"likes":{$lte:50}}) | where likes <= 50 |
| Greater Than | {<key> : {$gt:<value>}} | db.mycol.find( {"likes":{$gt:50}}) | where likes > 50 |
| Greater Than Equals | {<key> : {$gte:<value>}} | db.mycol.find( {"likes":{$gte:50}}) | where likes >= 50 |
| Not Equals | {<key> : {$ne:<value>}} | db.mycol.find( {"likes":{$ne:50}}) | where likes != 50 |

- Collection restaurants for query examples



- Query on a single field

```
> db.movie.find({imdb : "tt0084726"}).pretty()
{
        "_id" : ObjectId("59e92ad38eeffc9227fa4a82"),
        "imdb" : "tt0084726",
        "title" : "Star Trek II: The Wrath of Khan",
        "year" : 1982,
        "type" : "movie"
}
```

```
> db.movie.find({year : 2009}).pretty()
{
        "_id" : ObjectId("59e92e4d8eeffc9227fa4a83"),
        "imdb" : "tt0796366",
        "title" : "Star Trek",
        "year" : 2009,
        "type" : "movie"
}
{
        "_id" : ObjectId("59e94bca8eeffc9227fa4a8c"),
        "imdb" : "tt1401081",
        "title" : "Ice Age 3: Dawn of the Dinosaurs",
        "year" : 2009,
        "type" : "cartoon"
}
```

- Query on multiple fields
  - MongoDB treats each property as having an implicit boolean AND.

```
>db.restaurants.find({cuisine : "American", borough : "Bronx" }).limit(2)
```

```
> db.restaurants.find({cuisine : "American", borough : "Bronx" }).limit(2)
{ "_id" : ObjectId("59e565aa8b8edcdd5b7a43c5"), "address" : { "building" : "2300
", "coord" : [ -73.8786113, 40.8502883 ], "street" : "Southern Boulevard", "zipc
ode" : "10460" }, "borough" : "Bronx", "cuisine" : "American", "grades" : [ { "d
ate" : ISODate("2014-05-28T00:00:00Z"), "grade" : "A", "score" : 11 }, { "date"
: ISODate("2013-06-19T00:00:00Z"), "grade" : "A", "score" : 4 }, { "date" : ISOD
ate("2012-06-15T00:00:00Z"), "grade" : "A", "score" : 3 } ], "name" : "Wild Asia
", "restaurant_id" : "40357217" }
{ "_id" : ObjectId("59e565aa8b8edcdd5b7a43f6"), "address" : { "building" : "658"
, "coord" : [ -73.81363999999999, 40.82941100000001 ], "street" : "Clarence Ave"
, "zipcode" : "10465" }, "borough" : "Bronx", "cuisine" : "American", "grades" :
 [ { "date" : ISODate("2014-06-21T00:00:00Z"), "grade" : "A", "score" : 5 }, { "
date" : ISODate("2012-07-11T00:00:00Z"), "grade" : "A", "score" : 10 } ], "name"
 : "Manhem Club", "restaurant_id" : "40364363" }
```

- MongoDB supports boolean OR queries, but you must use a special operator ($or) to achieve it.

```
>db.restaurants.find({$or: [{restaurant_id : "30075445"}, {restaurant_id : "30191841"}]})
```

```
> db.restaurants.find({$or: [{restaurant_id : "30075445"}, {restaurant_id : "30191841"}]})
{ "_id" : ObjectId("59e565aa8b8edcdd5b7a43bb"), "address" : { "building" : "1007", "coord" : [ -73.856077, 40.848447 ], "street" : "Morris Park Ave", "zipcode" : "10462" }, "borough" : "Bronx", "cuisine" : "Bakery", "grades" : [ { "date" : ISODate("2014-03-03T00:00:00Z"), "grade" : "A", "score" : 2 }, { "date" : ISODate("2013-09-11T00:00:00Z"), "grade" : "A", "score" : 6 }, { "date" : ISODate("2013-01-24T00:00:00Z"), "grade" : "A", "score" : 10 }, { "date" : ISODate("2011-11-23T00:00:00Z"), "grade" : "A", "score" : 9 }, { "date" : ISODate("2011-03-10T00:00:00Z"), "grade" : "B", "score" : 14 } ], "name" : "Morris Park Bake Shop", "restaurant_id" : "30075445" }
{ "_id" : ObjectId("59e565aa8b8edcdd5b7a43bd"), "address" : { "building" : "351", "coord" : [ -73.98513559999999, 40.7676919 ], "street" : "West   57 Street", "zipcode" : "10019" }, "borough" : "Manhattan", "cuisine" : "Irish", "grades" : [ { "date" : ISODate("2014-09-06T00:00:00Z"), "grade" : "A", "score" : 2 }, { "date" : ISODate("2013-07-22T00:00:00Z"), "grade" : "A", "score" : 11 }, { "date" : ISODate("2012-07-31T00:00:00Z"), "grade" : "A", "score" : 12 }, { "date" : ISODate("2011-12-29T00:00:00Z"), "grade" : "A", "score" : 12 } ], "name" : "Dj Reynolds Pub And Restaurant", "restaurant_id" : "30191841" }
```

- *$gt* opertor (greater than) on a field value

```
>db.movie.find({year: {$gt:2005}})
```

```
> db.movie.find({year: {$gt:2005}})
{ "_id" : ObjectId("59e92e4d8eeffc9227fa4a83"), "imdb" : "tt0796366", "title" :
"Star Trek", "year" : 2009, "type" : "movie" }
{ "_id" : ObjectId("59e92e4d8eeffc9227fa4a84"), "imdb" : "tt1408101", "title" :
"Star Trek Into Darkness", "year" : 2013, "type" : "movie" }
{ "_id" : ObjectId("59e94bca8eeffc9227fa4a8b"), "imdb" : "tt0769366", "title" :
"Ice Age 2: The MEltdown", "year" : 2006, "type" : "cartoon" }
{ "_id" : ObjectId("59e94bca8eeffc9227fa4a8c"), "imdb" : "tt1401081", "title" :
"Ice Age 3: Dawn of the Dinosaurs", "year" : 2009, "type" : "cartoon" }
{ "_id" : ObjectId("59e94bca8eeffc9227fa4a8d"), "imdb" : "tt0171831", "title" :
"Ice Age 4: Continental Drift", "year" : 2012, "type" : "cartoon" }
```

*db.collection.find( <query filter>, <projection> ).<cursor modifier>*

*Like FROM clause, specifies the collection to use*

*Like WHERE clause, specifies which documents to return*

*Projection variables in SELECT clause*

*How many results to return etc.*

# The find() Function - projection

- Projection document specifies the fields to return in the documents that match the query filter.
- To return all fields in the matching documents, omit this parameter.
- Form of the projection document:

{ field1: <value>, field2: <value> ... }

The <value> can be any of the following:

- 1 or true to include the field in the return documents.
- 0 or false to exclude the field.

- Projection example:

```
>db.restaurants.find({cuisine : "American", borough : "Bronx" }, {borough :1, restaurant_id: 1}).limit(2).sort({restaurant_id : 1}).pretty()
```

```
> db.restaurants.find({cuisine : "American", borough : "Bronx" }, {borough :1, r
estaurant_id: 1}).limit(2).sort({restaurant_id : 1}).pretty()
{
        "_id" : ObjectId("59e565aa8b8edcdd5b7a43c5"),
        "borough" : "Bronx",
        "restaurant_id" : "40357217"
}
{
        "_id" : ObjectId("59e565aa8b8edcdd5b7a43f6"),
        "borough" : "Bronx",
        "restaurant_id" : "40364363"
}
>
```

- The _*id* field is included by default, need to exclude explicitly if you don't want it

```
>db.restaurants.find({cuisine : "American", borough : "Bronx" }, {borough :1, restaurant_id : 1, _id : 0}).limit(2).sort({restaurant_id : 1}).pretty()
```

```
> db.restaurants.find({cuisine : "American", borough : "Bronx" }, {borough :1, r
estaurant_id : 1, _id : 0}).limit(2).sort({restaurant_id : 1}).pretty()
{ "borough" : "Bronx", "restaurant_id" : "40357217" }
{ "borough" : "Bronx", "restaurant_id" : "40364363" }
>
```

# Summary

- JSON document
- MongoDB Document Structure
- Getting start with MongoDB
- MongoDB CRUD Operations
- Data Retrieval in MongoDB