Advanced Java Programming Course

Big Data - MongoDB

Faculty of Information Technologies
Industrial University of Ho Chi Minh City

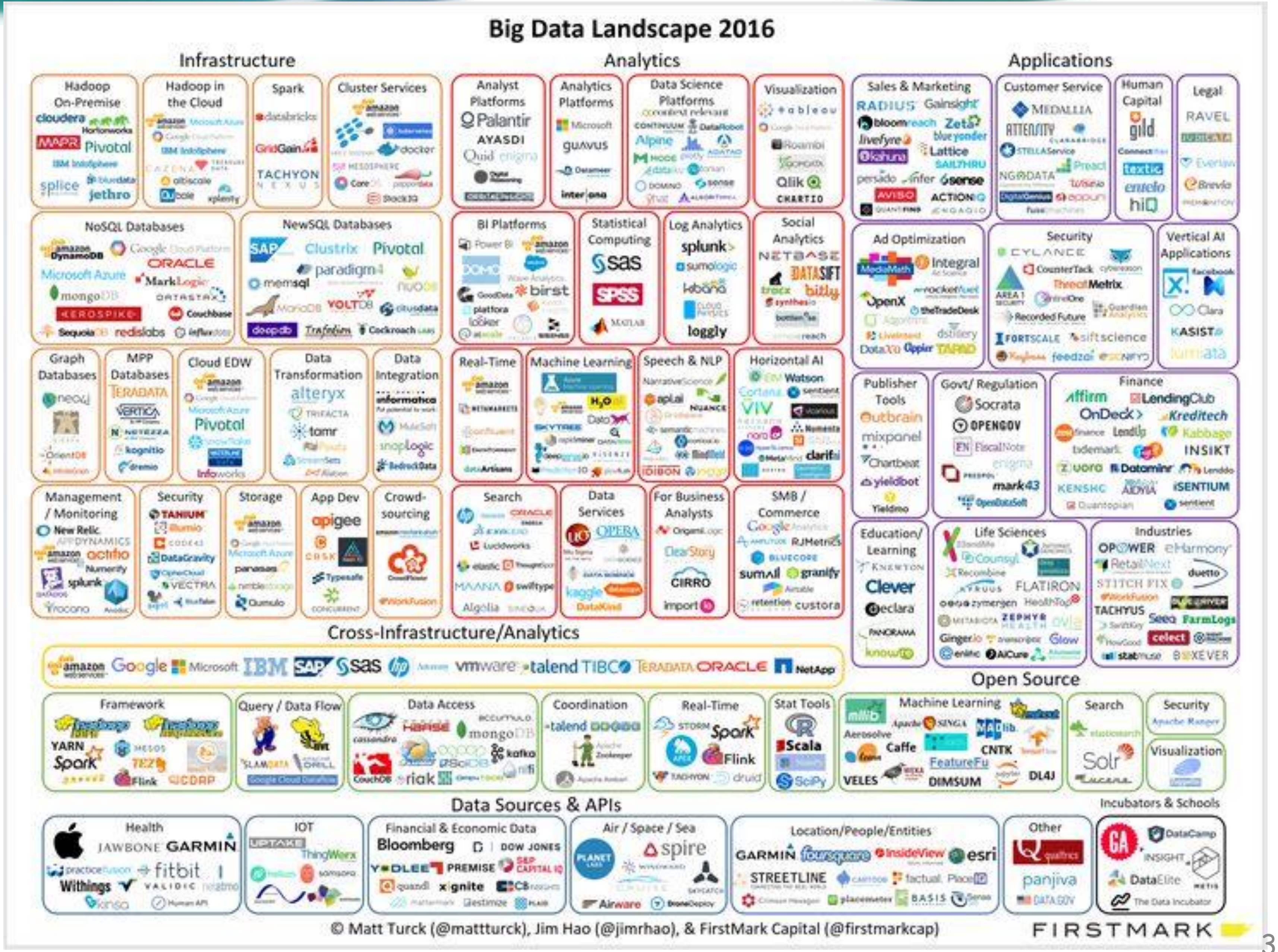
Session objectives

Big Data Overview

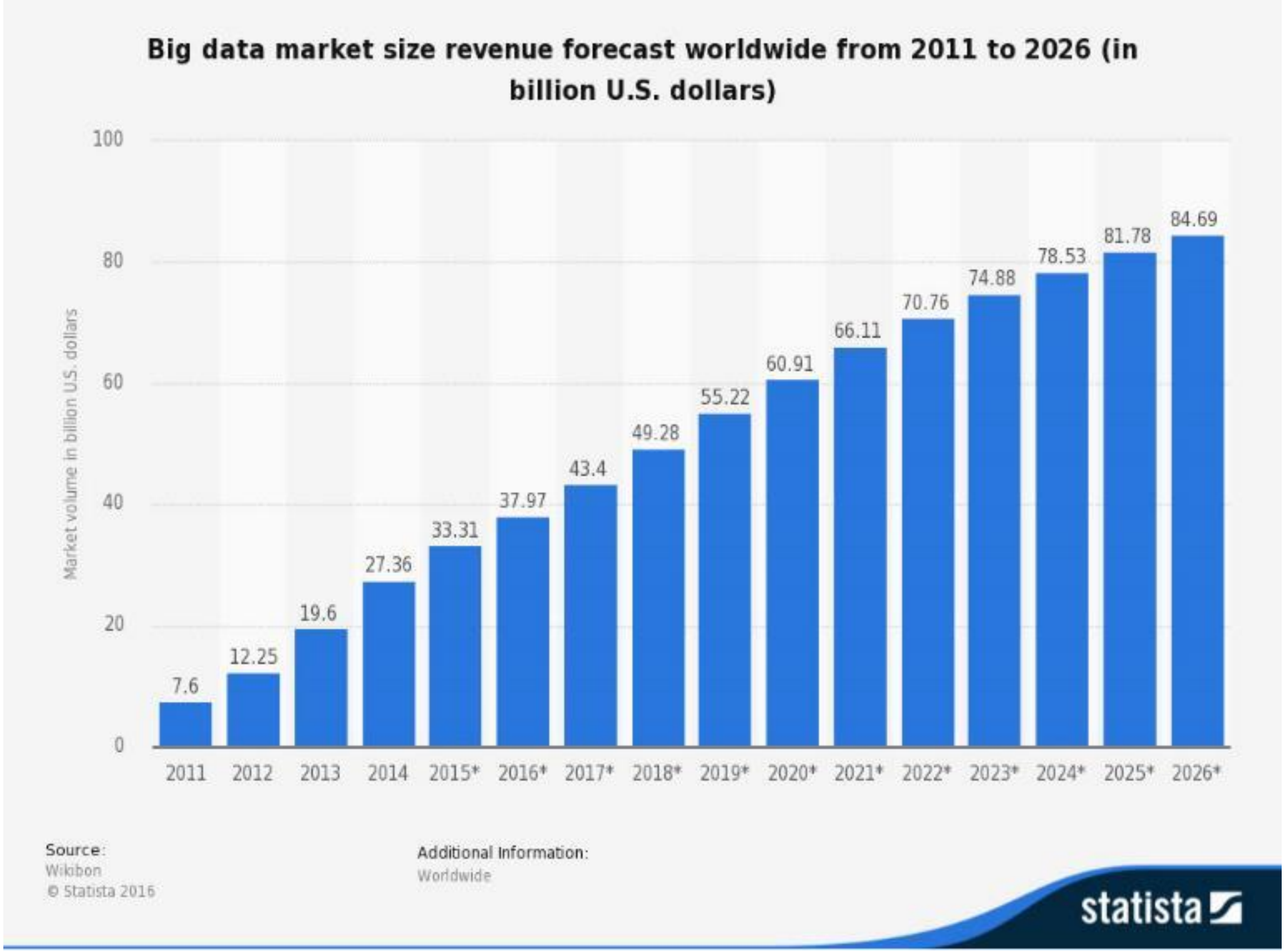
NoSQL introduction

MongoDB introduction

MongoDB – Java Programming


Big Data Landscape 2016
Infrastructure
Hadoop On-Premise
Hadoop in the Cloud
Spark
Cluster Services
NoSQL Databases
NewSQL Databases
Graph Databases
MPP Databases
Cloud EDW
Data Transformation
Data Integration
Management / Monitoring
Security
Storage
App Dev
Crowd-sourcing
Analytics
Analyst Platforms
Analytics Platforms
Data Science Platforms
Visualization
BI Platforms
Statistical Computing
Log Analytics
Social Analytics
Real-Time
Machine Learning
Speech & NLP
Horizontal AI
Search
Data Services
For Business Analysts
SMB / Commerce
Applications
Sales & Marketing
Customer Service
Human Capital
Legal
Ad Optimization
Security
Vertical AI Applications
Publisher Tools
Govt/ Regulation
Finance
Education/ Learning
Life Sciences
Industries
Cross-Infrastructure/Analytics
Open Source
Framework
Query / Data Flow
Data Access
Coordination
Real-Time
Stat Tools
Machine Learning
Search
Security
Visualization
Data Sources & APIs
Health
IOT
Financial & Economic Data
Air / Space / Sea
Location/People/Entities
Other
Incubators & Schools
© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark Capital (@firstmarkcap)
FIRSTMARK

Big Data, the market value



Big data market size revenue forecast worldwide from 2011 to 2026 (in billion U.S. dollars)

Source:
Wikibon
© Statista 2016

Additional Information:
Worldwide

Data Management Systems: History

- In the last decades RDBMS have been successful in solving problems related to storing, serving and processing data.
- RDBMS are adopted for:
  - Online transaction processing (OLTP),
  - Online analytical processing (OLAP).
- Vendors such as Oracle, Vertica, Teradata, Microsoft and IBM proposed their solution based on Relational Math and SQL.

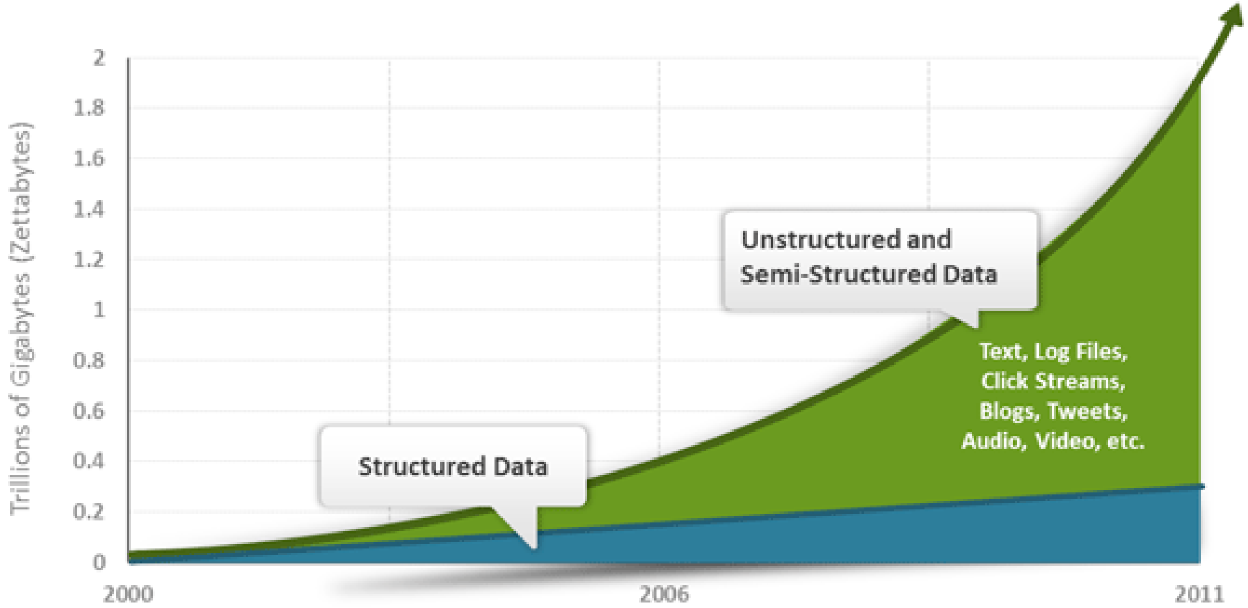
But….

Something Changed!

- Traditionally there were transaction recording (OLTP) and analytics (OLAP) of the recorded data.
- Not much was done to understand:
  - the reasons behind transactions,
  - what factor contributed to business, and
  - what factor could drive the customer's behavior.
- Pursuing such initiatives requires working with a large amount of varied data.

Something Changed!

- This approach was pioneered by Google, Amazon, Yahoo, Facebook and LinkedIn.
- They work with different type of data, often semi or un-structured.
- And they have to store, serve and process huge amount of data.



Source: IDC 2011 Digital Universe Study (http://www.emc.com/collateral/demos/microsites/emc-digital-universe-2011/index.htm)

Something Changed!

- RDBMS can somehow deal with this aspects, but they have issues related to:
  - expensive licensing,
  - requiring complex application logic,
  - Dealing with evolving data models
- There were a need for systems that could:
  - work with different kind of data format,
  - Do not require strict schema,
  - and are easily scalable.

Evolutions in Data Management

- As part of innovation in data management system, several new technologies where built:
  - 2003 - Google File System,
  - 2004 - MapReduce,
  - 2006 - BigTable,
  - 2007 - Amazon DynamoDB
  - 2012 - Google Cloud Engine
- Each solved different use cases and had a different set of assumptions.
- All these mark the beginning of a different way of thinking about data management.

Go to hell RDBMS!

Hello, Big Data!

Definition

"Big data is a term for data sets that are so large or complex that traditional data processing application software is inadequate to deal with them. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating and information privacy."

(https://en.wikipedia.org/wiki/Big_data)

Characteristics

Variety
Volume
Veracity
Velocity
Value

- Volume
  - The quantity of generated and stored data. The size of the data determines the value and potential insight- and whether it can actually be considered big data or not.
- Variety
  - The type and nature of the data. This helps people who analyze it to effectively use the resulting insight.
- Velocity
  - In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development.
- Variability
  - Inconsistency of the data set can hamper processes to handle and manage it.
- Veracity
  - The quality of captured data can vary greatly, affecting the accurate analysis.

NoSQL

NoSQL - history

- In 2006 Google published BigTable paper.
- In 2007 Amazon presented DynamoDB.
- It didn't take long for all these ideas to used in:
  - Several open source projects (Hbase, Cassandra) and
  - Other companies (Facebook, Twitter, …)
- And now? Now, nosql-database.org lists more than 225 NoSQL databases.

NoSQL related facts

- Explosion of social media sites (Facebook, Twitter) with large data needs.
- Rise of cloud-based solutions such as Amazon S3 (simple storage solution).
- Moving to dynamically-typed languages (Ruby/Groovy), a shift to dynamically-typed data with frequent schema changes.
- Functional Programming (Scala, Clojure, Erlang).

NoSQL Definition

http://nosql-database.org

“Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.

The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more. So the misleading term "nosql" (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above.”

NoSQL Categorization

1. Wide Column Store / Column Families
2. **Document Store**
3. Key Value / Tuple Store
4. Graph Databases
5. Multimodel Databases
6. Object Databases
7. Grid & Cloud Database Solutions
8. XML Databases
9. Multidimensional Databases
10. Multivalue Databases
11. Event Sourcing
12. Time Series / Streaming Databases
13. Other NoSQL related databases
14. unresolved and uncategorized

Source: http://nosql-database.org

Key Value Store

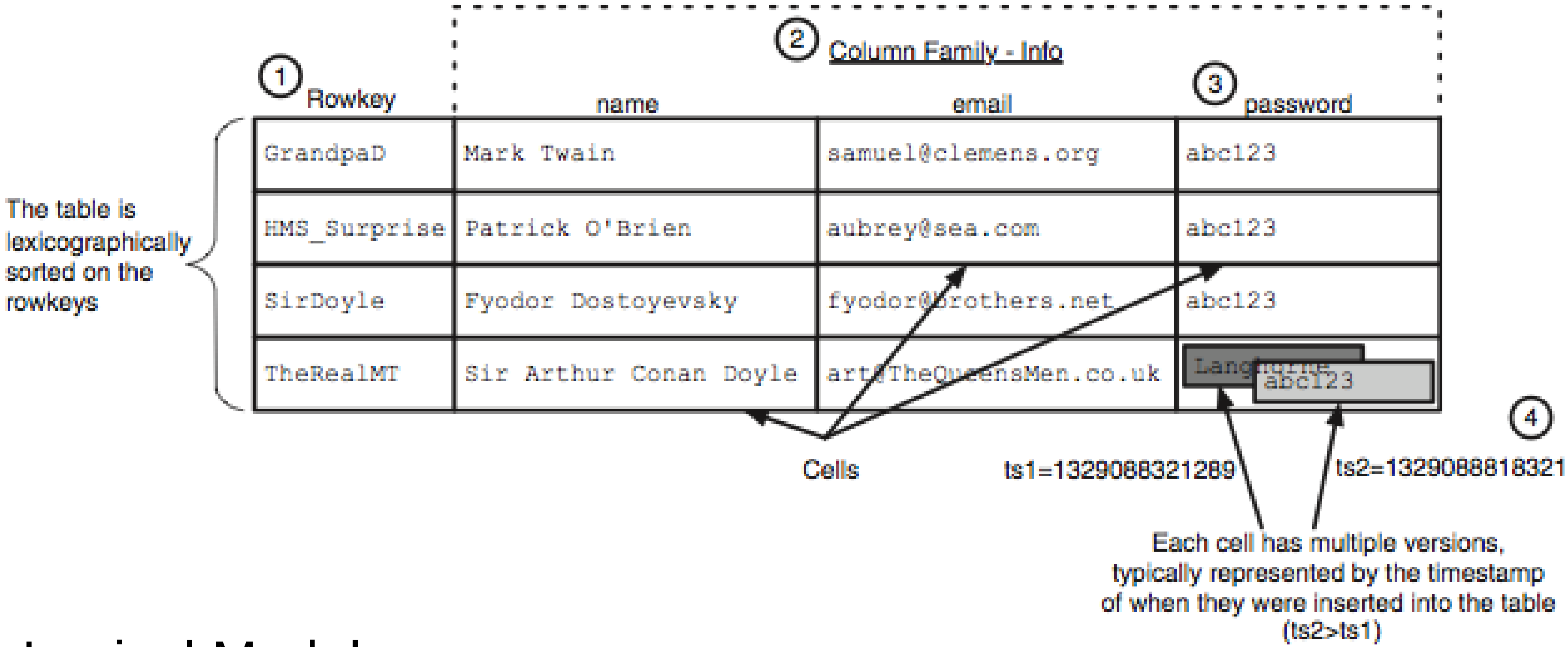
- Extremely simple interface:
  - Data model: (key, value) pairs
  - Basic Operations: : Insert(key, value), Fetch(key),Update(key), Delete(key)
- Values are store as a “blob”:
  - Without caring or knowing what is inside
  - The application layer has to understand the data
- Advantages: efficiency, scalability, fault-tolerance

- Pros:
  - very fast
  - very scalable
  - simple model
  - able to distribute horizontally
- Cons:
  - many data structures (objects) can't be easily modeled as key value pairs

Column-oriented (1)

- Store data in columnar format
- Each storage block contains data from only one column
- Allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - data model: families of attributes defined in a schema, new attributes can be added online
  - storing principle: big hashed distributed tables
  - properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application

Column-oriented (2)



Rowkey	name	email	password
GrandpaD	Mark Twain	samuel@clemens.org	abc123
HMS_Surprise	Patrick O'Brien	aubrey@sea.com	abc123
SirDoyle	Fyodor Dostoyevsky	fyodor@brothers.net	abc123
TheRealMT	Sir Arthur Conan Doyle	art@TheQueensMen.co.uk	Langhorne abc123

The table is lexicographically sorted on the rowkeys

Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table (ts2>ts1)

Logical Model

Map<RowKey, Map<ColumnFamily, Map<ColumnQualifier, Map<Version, Data>>>>

Document Store

- Schema Free.
- Usually JSON (BSON) like interchange model, which supports lists, maps, dates, Boolean with nesting
- Query Model: JavaScript or custom.
- *Aggregations: Map/Reduce.*
- Indexes are done via B-Trees.
- Example: Mongo
  - {Name:"Jaroslav",

    Address:"Malostranske nám. 25, 118 00 Praha 1“

    Grandchildren: [Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1"]

    }

Document Store: Advantages

- Documents are independent units
- Application logic is easier to write. (JSON).
- Schema Free:
  - Unstructured data can be stored easily, since a document contains whatever keys and values the application logic requires.
  - In addition, costly migrations are avoided since the database does not need to know its information schema in advance.

Graph Databases

- They are significantly different from the other three classes of NoSQL databases.
- Graph Databases are based on the mathematical concept of graph theory.
- They fit well in several real world applications (twits, permission models)
- Are based on the concepts of Vertex and Edges
- A Graph DB can be labeled, directed, attributed multi-graph
- Relational DBs can model graphs, but an edge does not require a join which is expensive.

NoSQL: How to

https://en.wikipedia.org/wiki/CAP_theorem

http://www.julianbrowne.com/article/viewer/brewers-cap-theorem

https://dzone.com/articles/better-explaining-cap-theorem

Brewer's CAP Theorem

A distributed system can support only two of the following characteristics:

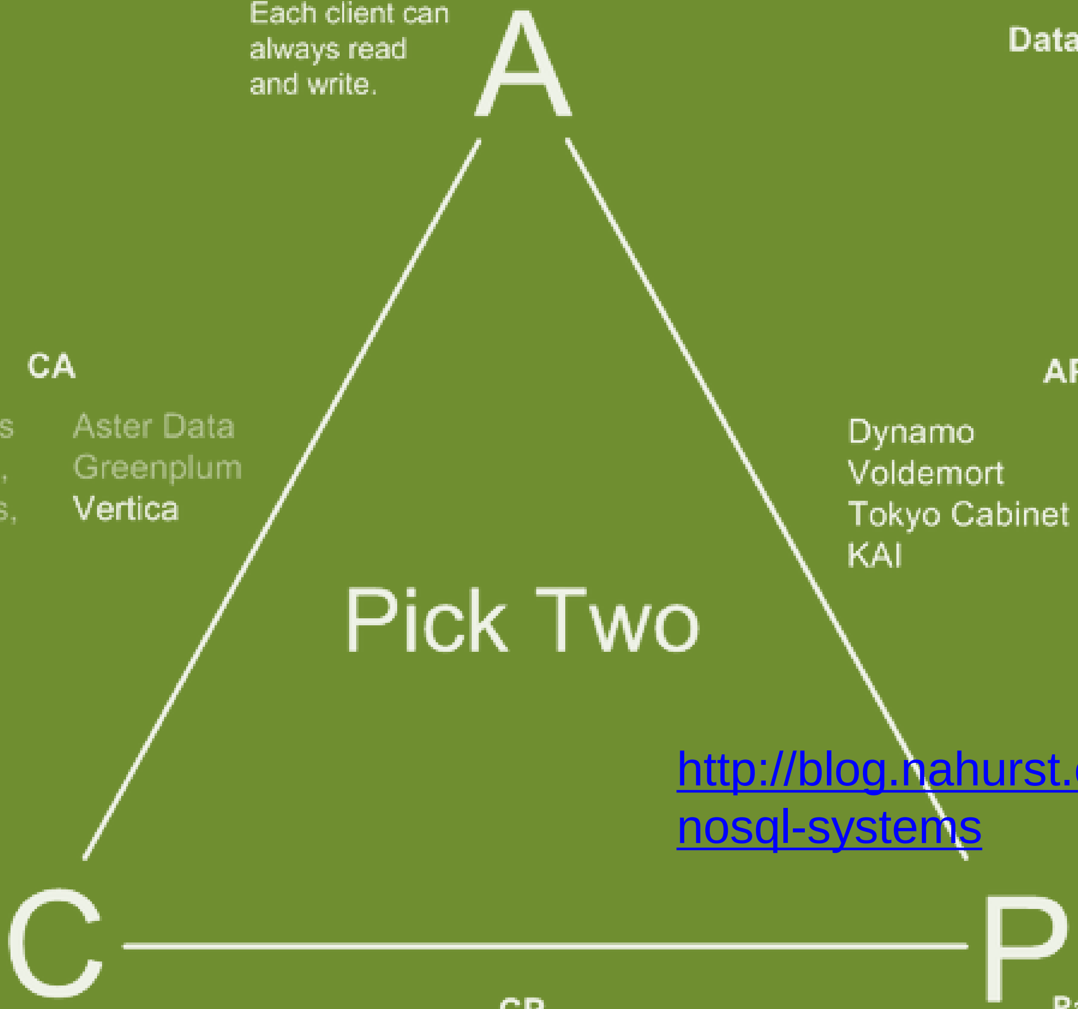
- Consistency (all copies have same value)
- Availability (system can run even if parts have failed)
- Partition Tolerance (network can break into two or more parts, each with active systems that can not influence other parts)

Brewer's CAP Theorem

Very large systems will partition at some point:

- it is necessary to decide between Consistency and Availability,
- traditional DBMS prefer Consistency over Availability and Partition,
- most Web applications choose Availability (except in specific applications such as order processing)

Visual Guide to NoSQL Systems



Availability:
Each client can
always read
and write.

Data Models

- Relational (comparison)
- Key-Value
- Column-Oriented/Tabular
- Document-Oriented

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

http://blog.nahurst.com/visual-guide-to-nosql-systems

Consistency:
All clients always
have the same view
of the data.

CP

BigTable
Hypertable
Hbase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

Partition Tolerance:
The system works
well despite physical
network partitions.

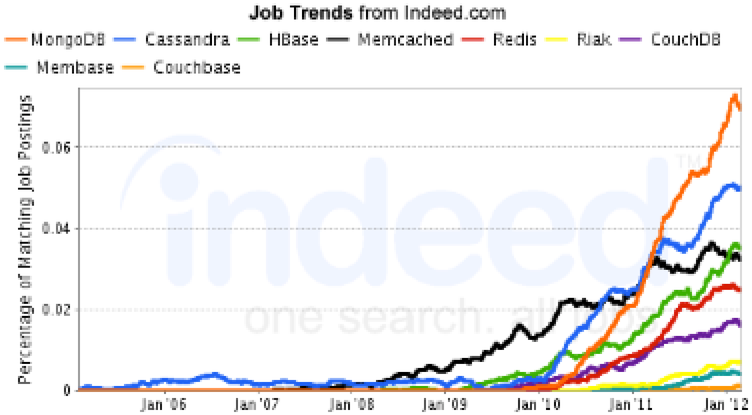
MongoDB

Introduction

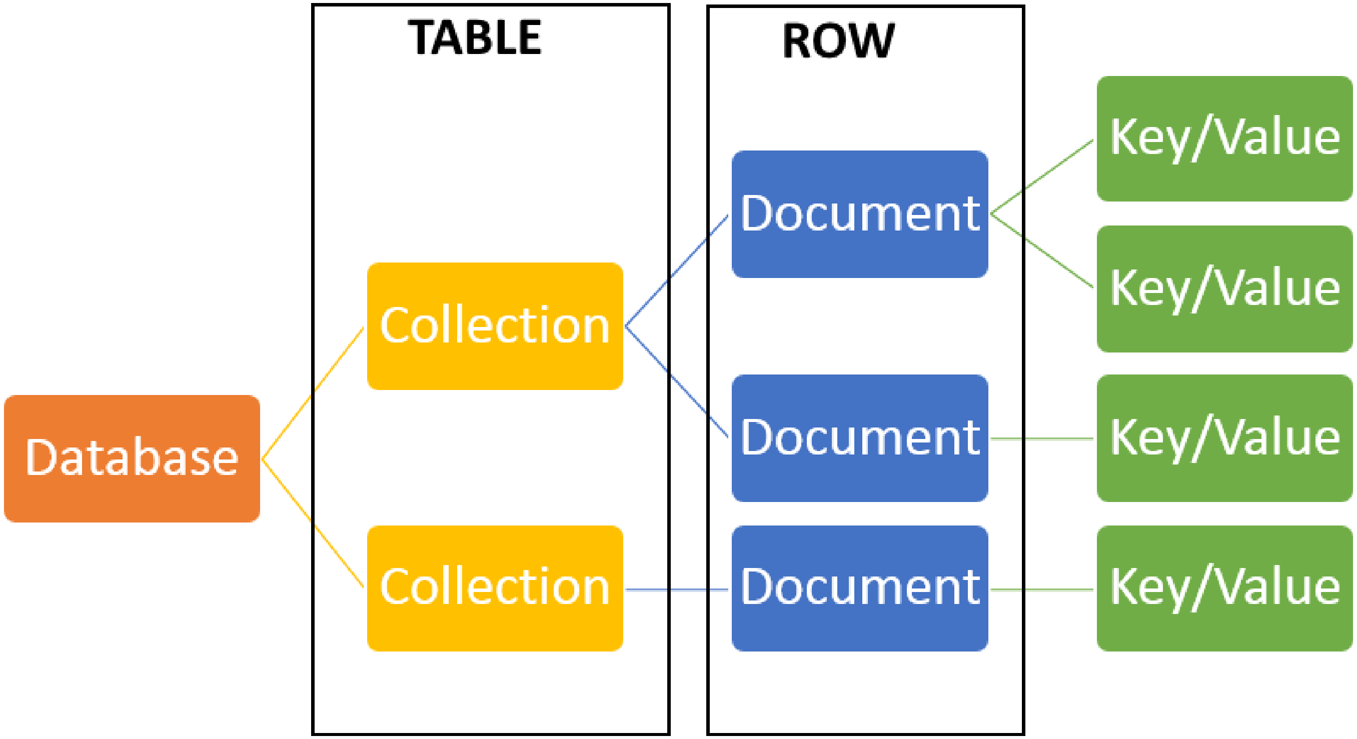
- MongoDB is an open-source database developed by MongoDB, Inc. (https://www.mongodb.com)
- MongoDB stores data in JSON-like (BSON) documents that can vary in structure.
- Related information is stored together for fast query access through the MongoDB query language.
- MongoDB uses dynamic schemas.

History

- 2007 - First developed (by 10gen)
- 2009 - Become Open Source
- 2010 - Considered production ready (v 1.4 >)
- *2013 - MongoDB Closes $150 Million in Funding*
- 2014 - Latest stable version (v 2.6)
- *Today- More than $231 million in total investment since 2007*
- *MongoDB inc. valuated $1.2B.*



MongoDB structure



Terminology and Concepts

SQL Terms/Concepts	MongoDB Terms/Concepts
database	database
table	collection
row	document or BSON document
column	field
index	index
table joins	$lookup, embedded documents
primary key Specify any unique column or column combination as primary key.	primary key In MongoDB, the primary key is automatically set to the _id field.
aggregation (e.g. group by)	aggregation pipeline

SQL to Aggregation Mapping Chart

SQL Terms, Functions, and Concepts	MongoDB Aggregation Operators
WHERE	$match
GROUP BY	$group
HAVING	$match
SELECT	$project
ORDER BY	$sort
LIMIT	$limit
SUM()	$sum
COUNT()	$sum
join	$lookup

MongoDB - Advantages

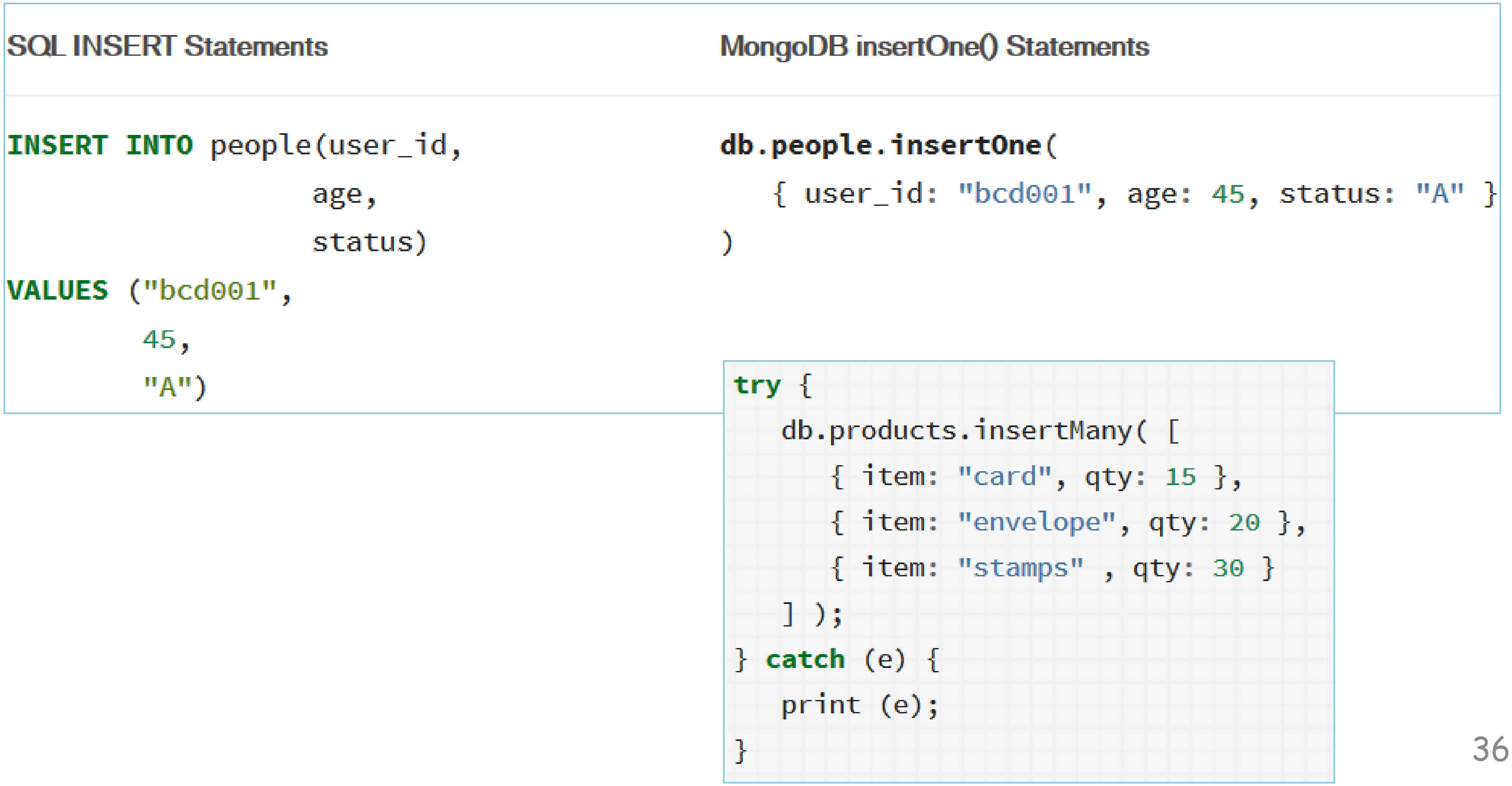
- Flexible Data Model
- Expressive Query Syntax
- Easy to Learn
- Performance
- Scalable and Reliable
- Async Drivers
- Documentation
- Text Search
- Server-Side Script
- Documents = Objects

MongoDB – The bad

- Transactions
- No Triggers
- More Storage
- Not automatically disk cleanup
- Hierarchy of Self
- Joins
- Indexing
- Duplicate Data

Insert document

- db.collection.insertOne()
- db.collection.insertMany()

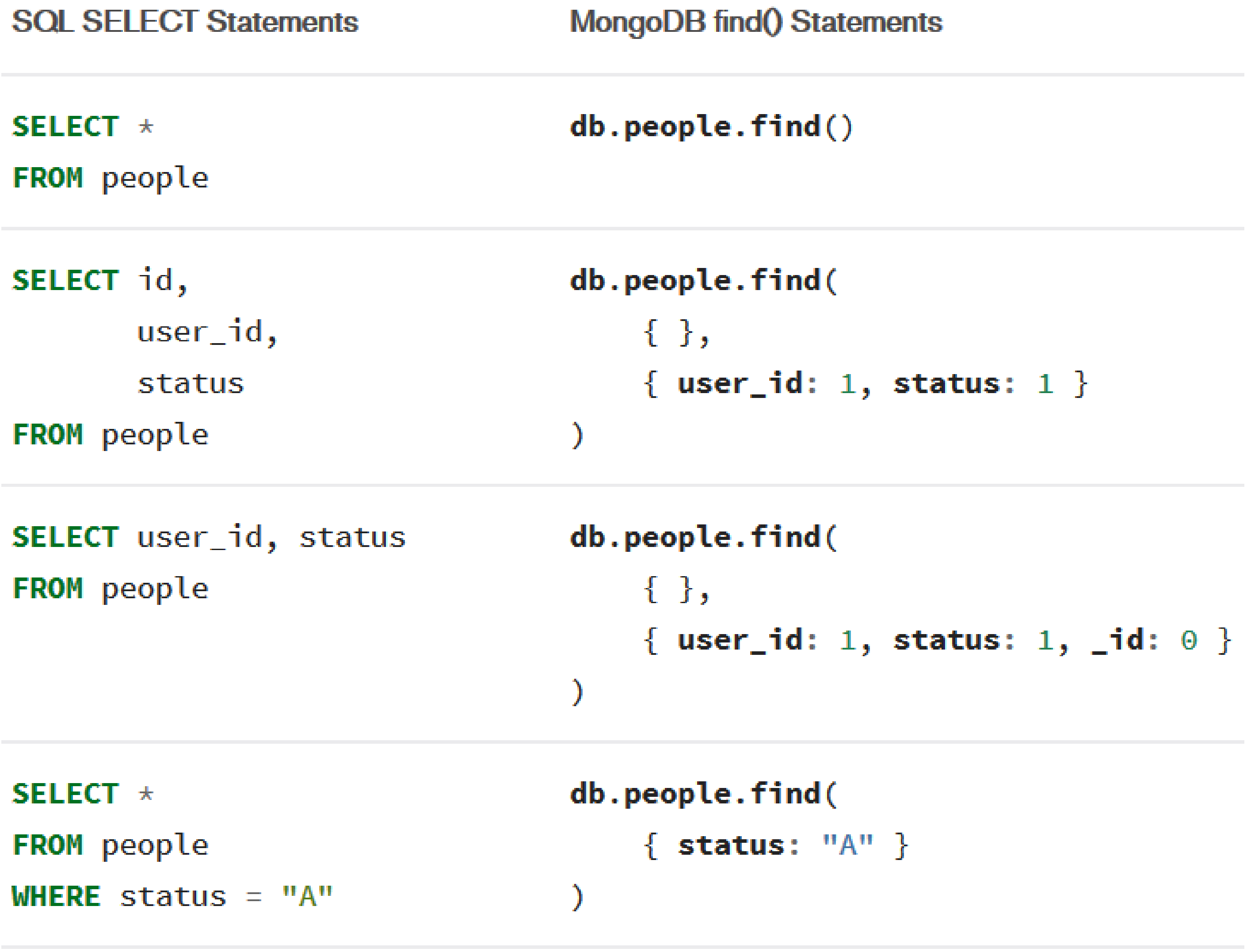


SQL INSERT Statements	MongoDB insertOne() Statements
INSERT INTO people(user_id, age, status) VALUES ("bcd001", 45, "A")	db.people.insertOne({ user_id: "bcd001", age: 45, status: "A" })

```
try {
   db.products.insertMany( [
      { item: "card", qty: 15 },
      { item: "envelope", qty: 20 },
      { item: "stamps" , qty: 30 }
   ] );
} catch (e) {
   print (e);
}
```

Find document(s)

db.collection.find(query, projection)



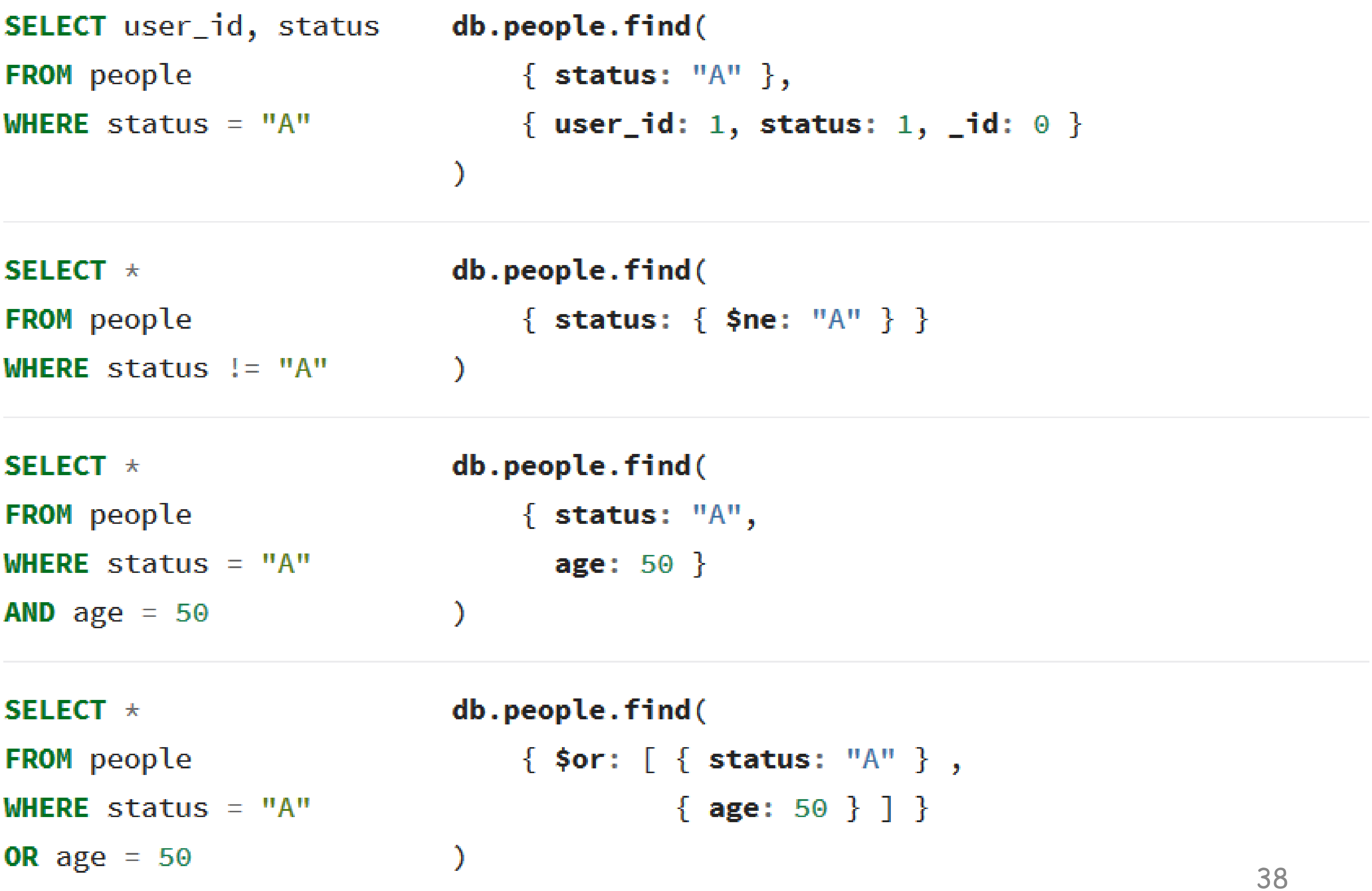
SQL SELECT Statements	MongoDB find() Statements

SELECT *
FROM people
	db.people.find()

SELECT id,
       user_id,
       status
FROM people
	db.people.find(
	    { },
	    { user_id: 1, status: 1 }
	)

SELECT user_id, status
FROM people
	db.people.find(
	    { },
	    { user_id: 1, status: 1, _id: 0 }
	)

SELECT *
FROM people
WHERE status = "A"
	db.people.find(
	    { status: "A" }
	)



```
SELECT user_id, status
FROM people
WHERE status = "A"
```

```
db.people.find(
    { status: "A" },
    { user_id: 1, status: 1, _id: 0 }
)
```

```
SELECT *
FROM people
WHERE status != "A"
```

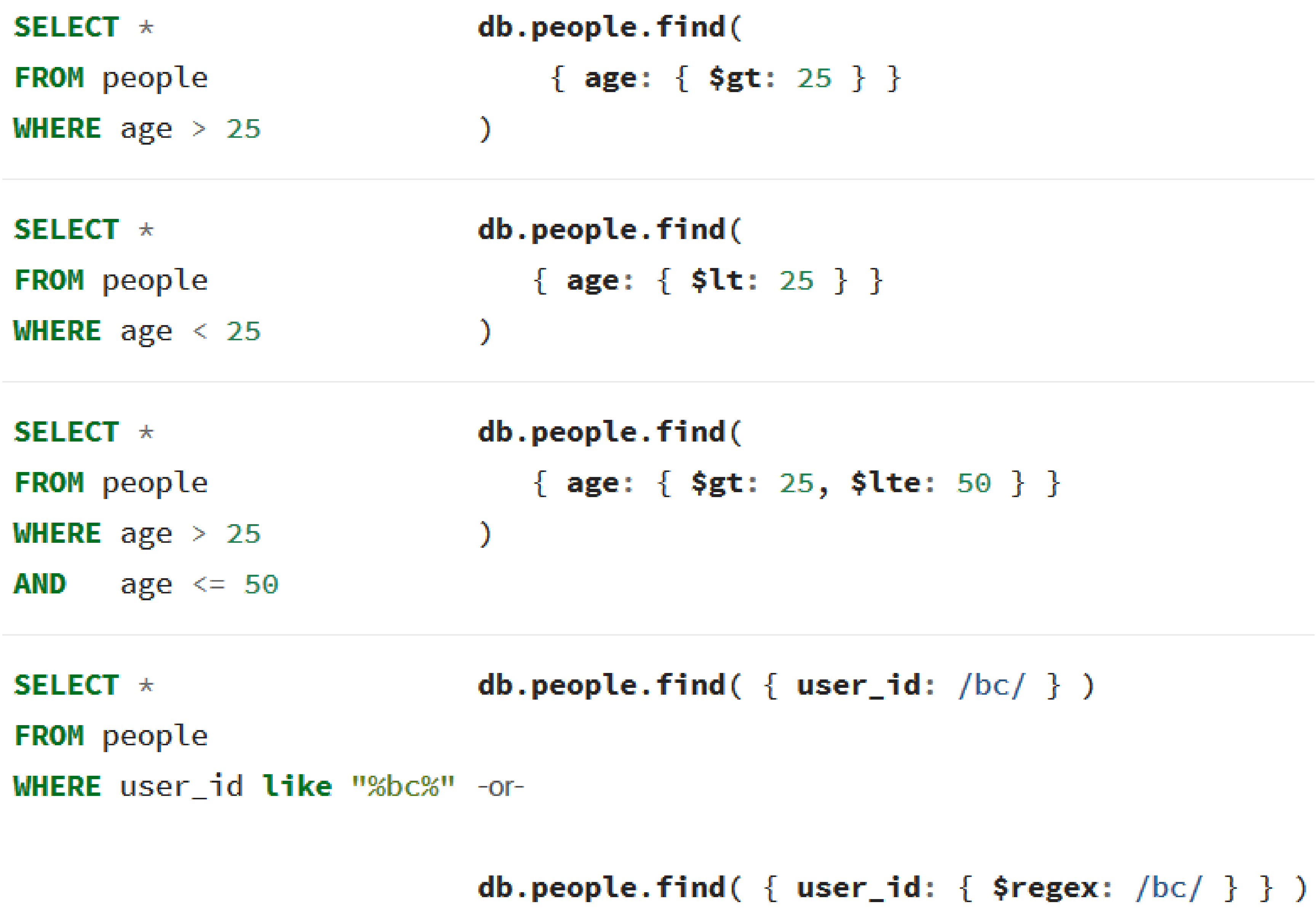
```
db.people.find(
    { status: { $ne: "A" } }
)
```

```
SELECT *
FROM people
WHERE status = "A"
AND age = 50
```

```
db.people.find(
    { status: "A",
      age: 50 }
)
```

```
SELECT *
FROM people
WHERE status = "A"
OR age = 50
```

```
db.people.find(
    { $or: [ { status: "A" } ,
             { age: 50 } ] }
)
```



```
SELECT *
FROM people
WHERE age > 25
```

```
db.people.find(
    { age: { $gt: 25 } }
)
```

```
SELECT *
FROM people
WHERE age < 25
```

```
db.people.find(
   { age: { $lt: 25 } }
)
```

```
SELECT *
FROM people
WHERE age > 25
AND   age <= 50
```

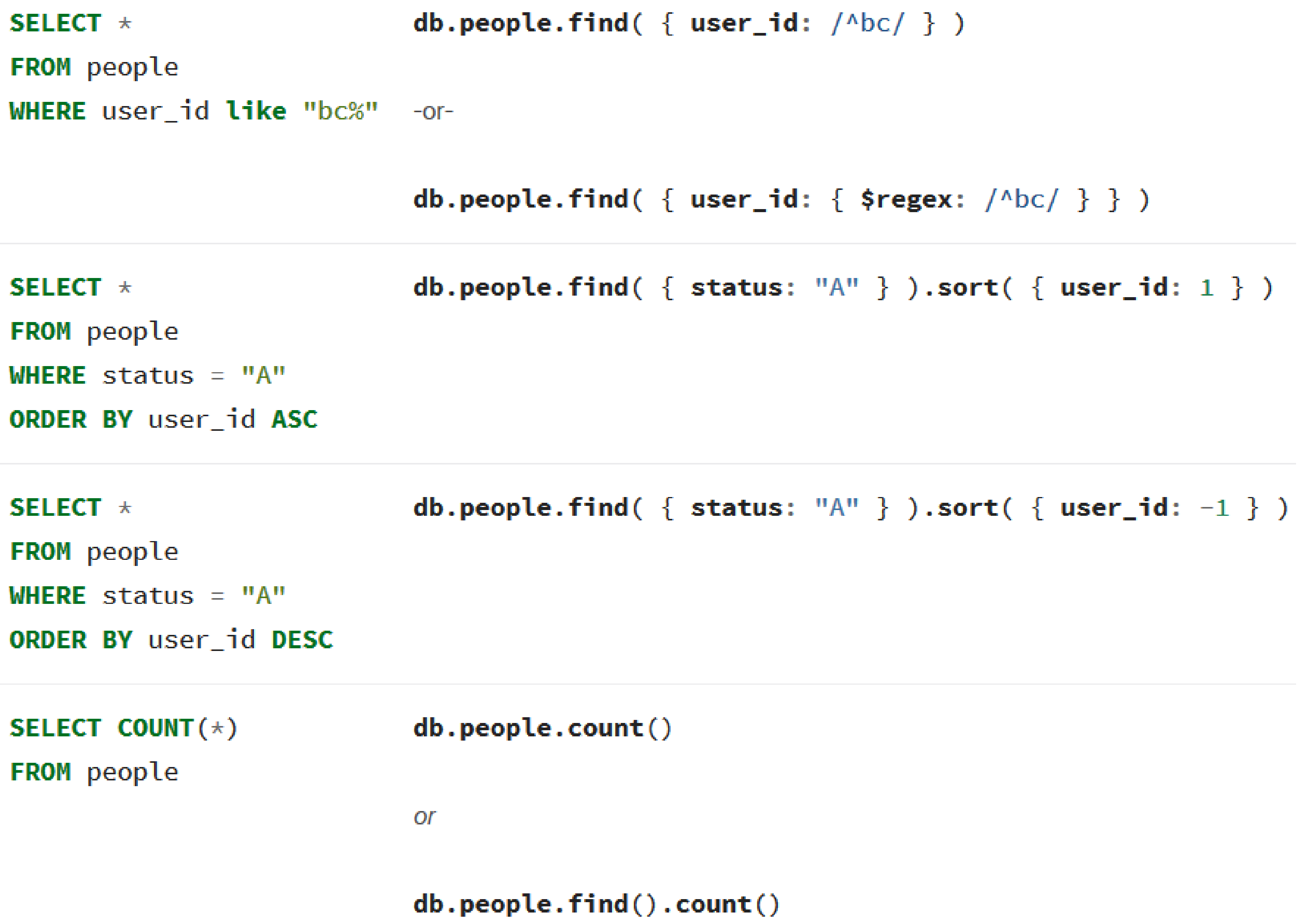
```
db.people.find(
   { age: { $gt: 25, $lte: 50 } }
)
```

```
SELECT *
FROM people
WHERE user_id like "%bc%"
```

```
db.people.find( { user_id: /bc/ } )
```

-or-

```
db.people.find( { user_id: { $regex: /bc/ } } )
```

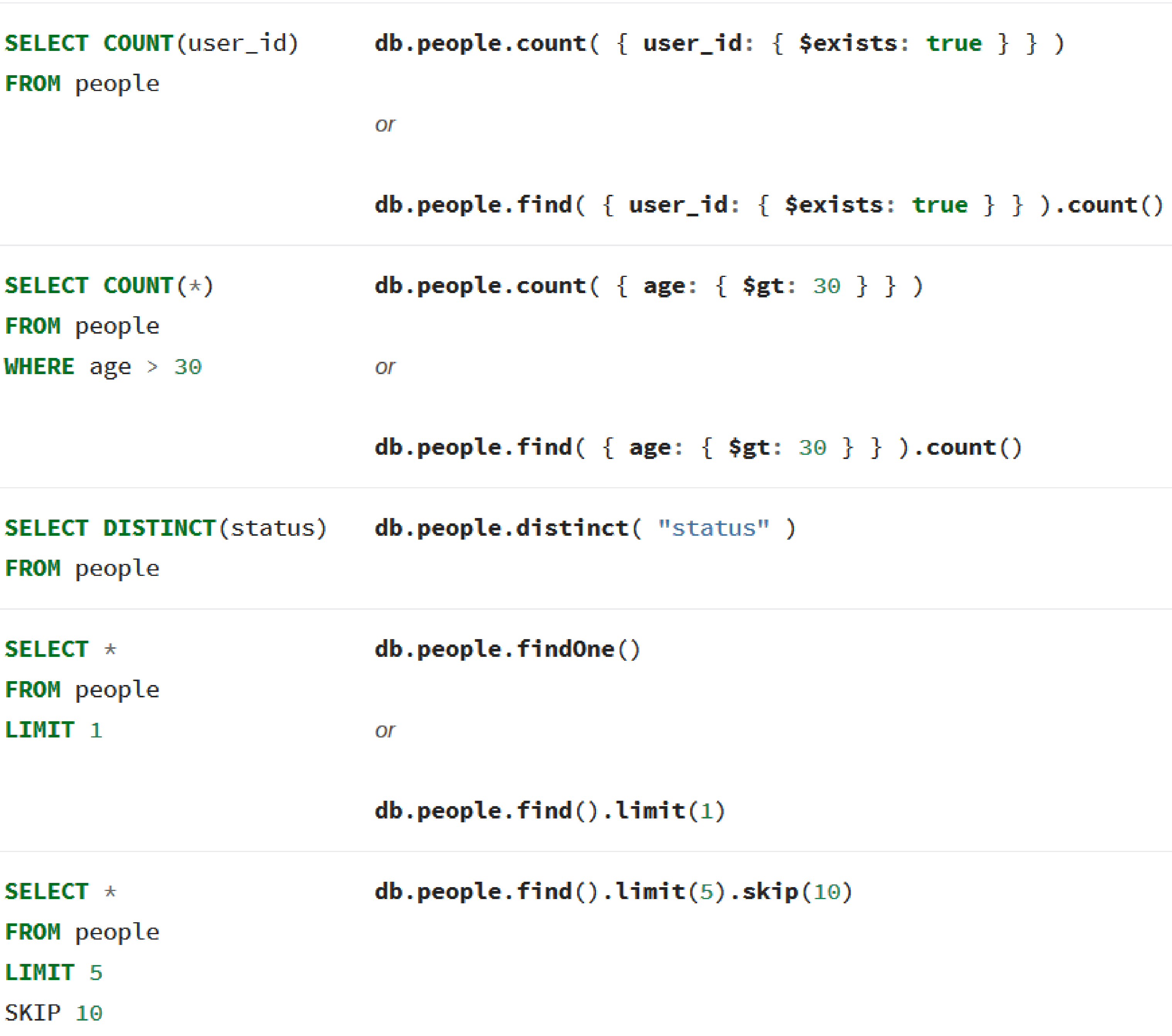


SELECT *
FROM people
WHERE user_id like "bc%"
	db.people.find({ user_id: /^bc/ })
	-or-
	db.people.find({ user_id: { $regex: /^bc/ } })

SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id ASC
	db.people.find({ status: "A" }).sort({ user_id: 1 })

SELECT *
FROM people
WHERE status = "A"
ORDER BY user_id DESC
	db.people.find({ status: "A" }).sort({ user_id: -1 })

SELECT COUNT(*)
FROM people
	db.people.count()
	or
	db.people.find().count()



`SELECT COUNT(user_id)`
`FROM people`
	`db.people.count( { user_id: { $exists: true } } )`
	or
	`db.people.find( { user_id: { $exists: true } } ).count()`

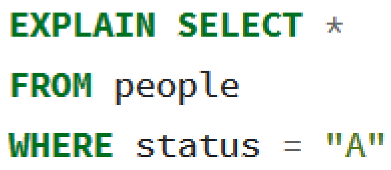
`SELECT COUNT(*)`
`FROM people`
`WHERE age > 30`
	`db.people.count( { age: { $gt: 30 } } )`
	or
	`db.people.find( { age: { $gt: 30 } } ).count()`

`SELECT DISTINCT(status)`
`FROM people`
	`db.people.distinct( "status" )`

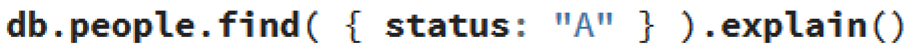
`SELECT *`
`FROM people`
`LIMIT 1`
	`db.people.findOne()`
	or
	`db.people.find().limit(1)`

`SELECT *`
`FROM people`
`LIMIT 5`
`SKIP 10`
	`db.people.find().limit(5).skip(10)`

Explain query



```
EXPLAIN SELECT *
FROM people
WHERE status = "A"
```



```
db.people.find( { status: "A" } ).explain()
```

Others criteria

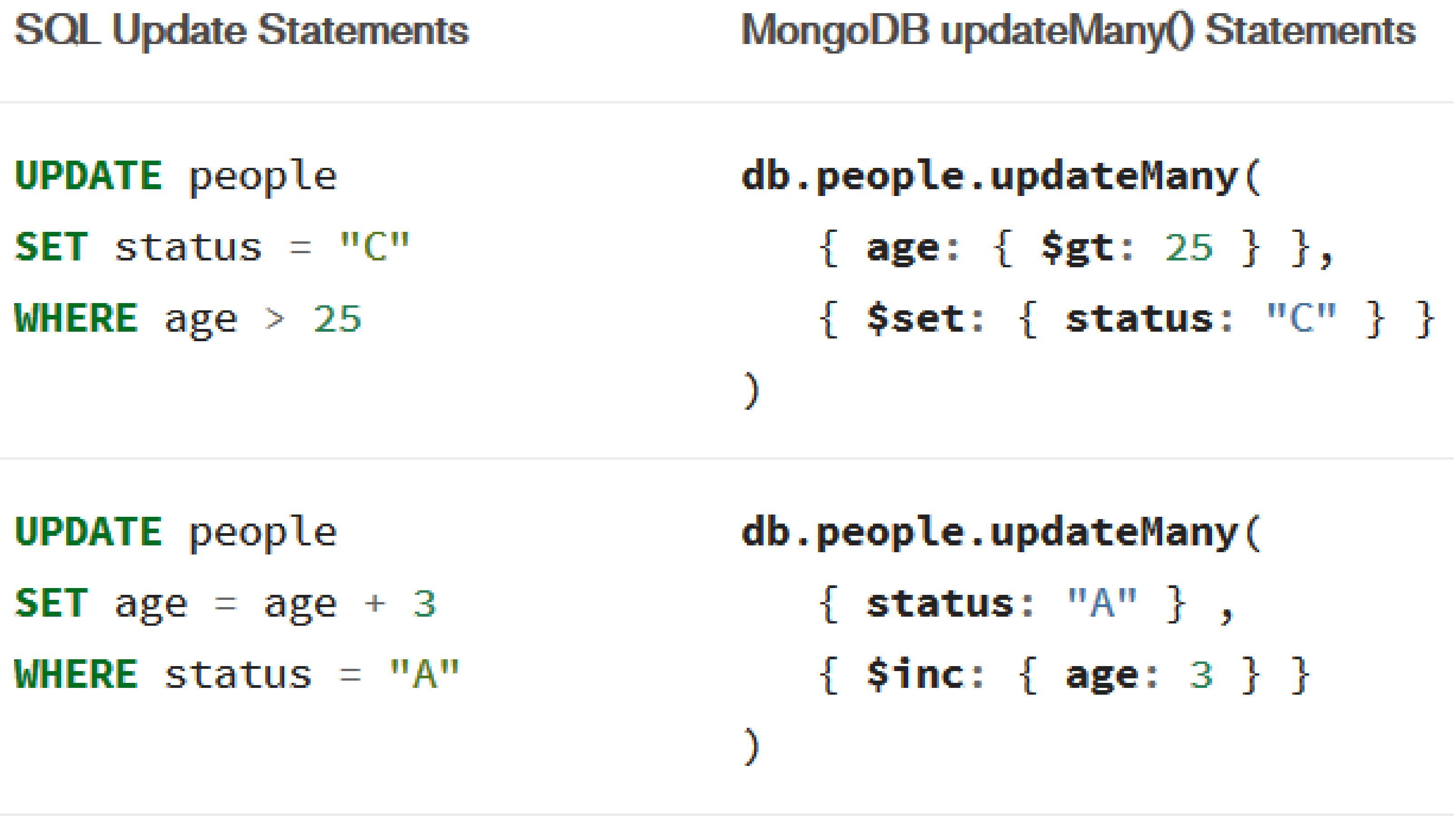
- limit()
- skip()
- explain()
- sort()
- count()
- pretty()
- …

Update document

db.collection.updateOne(<filter>, <update>, <options>)

db.collection.updateMany(<filter>, <update>, <options>)

db.collection.replaceOne(<filter>, <replacement>, <options>)



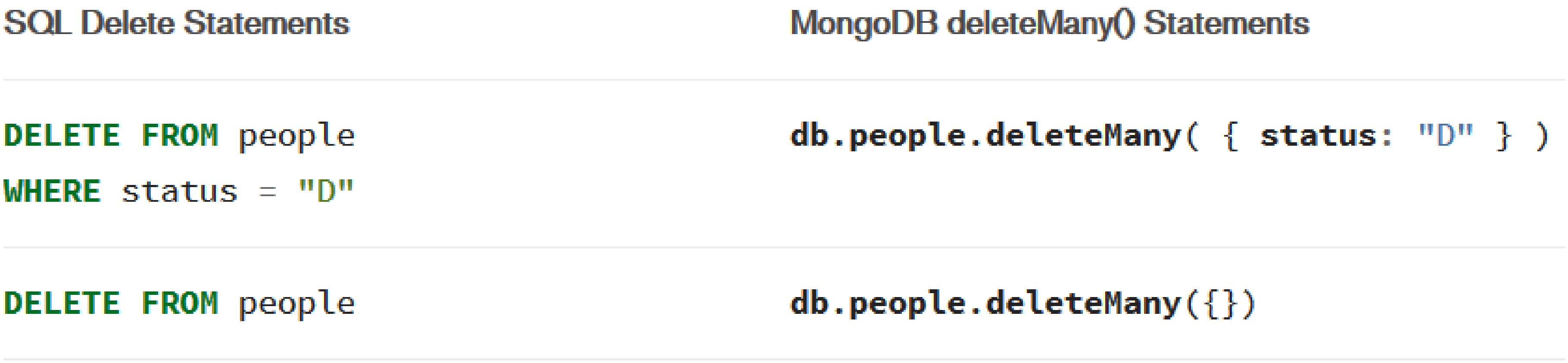
SQL Update Statements	MongoDB updateMany() Statements

`UPDATE people`
`SET status = "C"`
`WHERE age > 25`
	`db.people.updateMany(`
	`{ age: { $gt: 25 } },`
	`{ $set: { status: "C" } }`
	`)`

`UPDATE people`
`SET age = age + 3`
`WHERE status = "A"`
	`db.people.updateMany(`
	`{ status: "A" } ,`
	`{ $inc: { age: 3 } }`
	`)`

Delete document

- db.collection.deleteMany()
- db.collection.deleteOne()



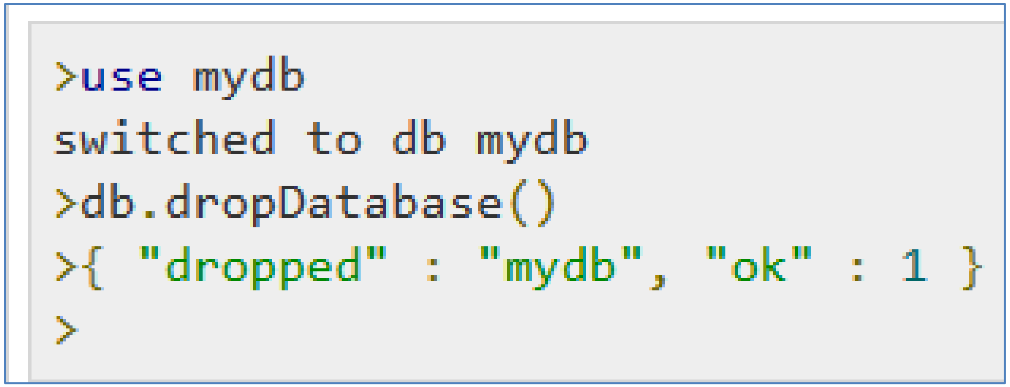
SQL Delete Statements	MongoDB deleteMany() Statements

`DELETE FROM people`
`WHERE status = "D"`
	`db.people.deleteMany( { status: "D" } )`

`DELETE FROM people`	`db.people.deleteMany({})`

Drop databse

- MongoDB db.dropDatabase() command is used to drop a existing database.
- This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.



```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

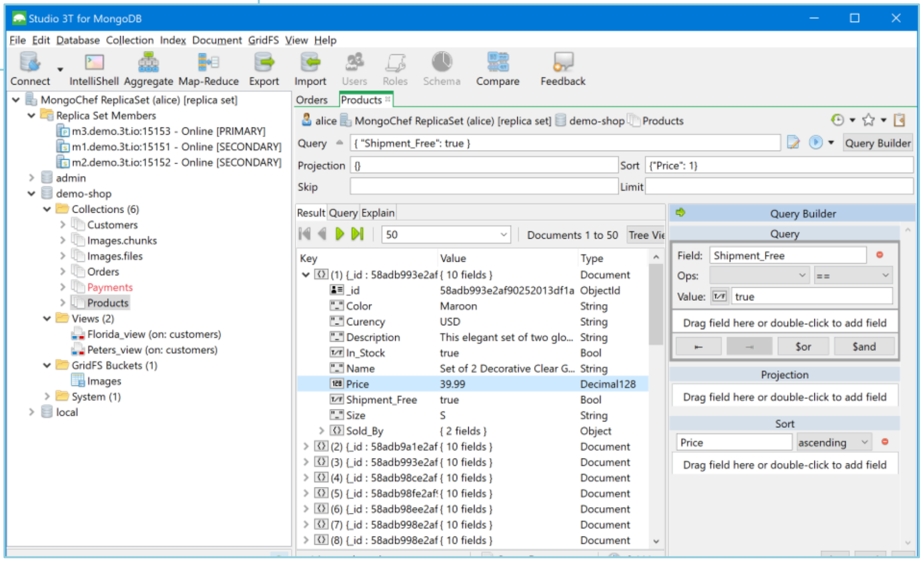
Using Management tools

MongoDB Compass

For fast schema discovery and visual construction of ad-hoc queries

- Visualize schema
  - Frequency of fields
  - Frequency of types
  - Determine validator rules
- View Documents
- Graphically build queries
- Authenticated access

...



```
MongoChef ReplicaSet (alice) [replica set]
  Replica Set Members
    m3.demo.3t.io:15153 - Online [PRIMARY]
    m1.demo.3t.io:15151 - Online [SECONDARY]
    m2.demo.3t.io:15152 - Online [SECONDARY]
  admin
  demo-shop
    Collections (6)
      Customers
      Images.chunks
      Images.files
      Orders
      Payments
      Products
    Views (2)
      Florida_view (on: customers)
      Peters_view (on: customers)
    GridFS Buckets (1)
      Images
    System (1)
  local
```

alice MongoChef ReplicaSet (alice) [replica set] demo-shop Products

Query: { "Shipment_Free": true }

Projection: {}

Sort: {"Price": 1}

Skip:

Limit:

50 Documents 1 to 50

```
Key                              Value                         Type
(1) {_id : 58adb993e2af          { 10 fields }                 Document
    _id                          58adb993e2af90252013df1a      ObjectId
    Color                        Maroon                        String
    Curency                      USD                           String
    Description                  This elegant set of two glo...  String
    In_Stock                     true                          Bool
    Name                         Set of 2 Decorative Clear G...  String
    Price                        39.99                         Decimal128
    Shipment_Free                true                          Bool
    Size                         S                             String
    Sold_By                      { 2 fields }                  Object
(2) {_id : 58adb9a1e2af          { 10 fields }                 Document
(3) {_id : 58adb993e2af          { 10 fields }                 Document
(4) {_id : 58adb98ce2af          { 10 fields }                 Document
(5) {_id : 58adb98fe2af          { 10 fields }                 Document
(6) {_id : 58adb98ee2af          { 10 fields }                 Document
(7) {_id : 58adb998e2af          { 10 fields }                 Document
(8) {_id : 58adb998e2af          { 10 fields }                 Document
```

Field: Shipment_Free

Ops: ==

Value: true

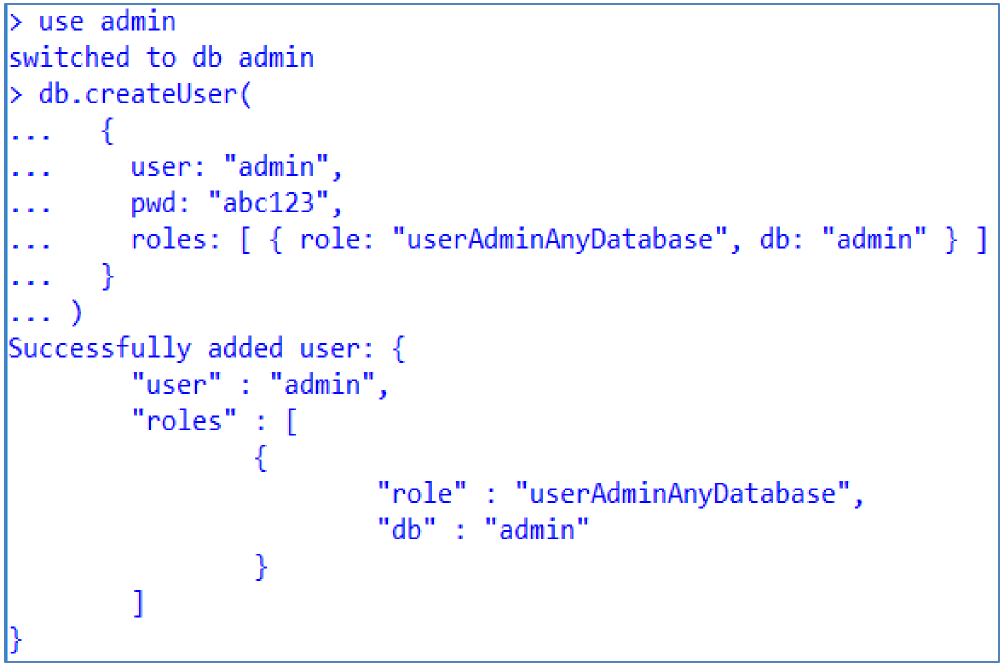
Sort: Price ascending

Authentication enable

- Grant permission to users to authenticate
  - Central database
  - Each database
- Policies:
  - readAnyDatabase
  - readWriteAnyDatabase
  - userAdminAnyDatabase
  - dbAdminAnyDatabase

Authentication enable

1. Create admin database
2. Add admin user



```
> use admin
switched to db admin
> db.createUser(
...    {
...      user: "admin",
...      pwd: "abc123",
...      roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
...    }
... )
Successfully added user: {
        "user" : "admin",
        "roles" : [
                {
                        "role" : "userAdminAnyDatabase",
                        "db" : "admin"
                }
        ]
}
```

3. Client logon:

   mongo -u "admin" -p "abc123" -authenticationDatabase "admin"

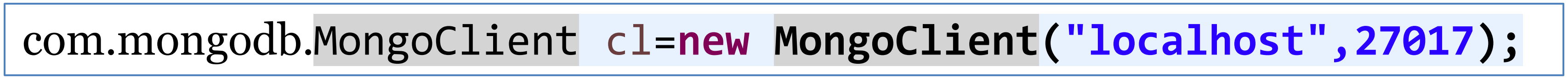
- Driver:

  http://mongodb.github.io/mongo-java-driver/
- Sync

  http://mongodb.github.io/mongo-java-driver/3.5/driver/
- A-Sync
  - http://mongodb.github.io/mongo-java-driver/3.5/driver-async/

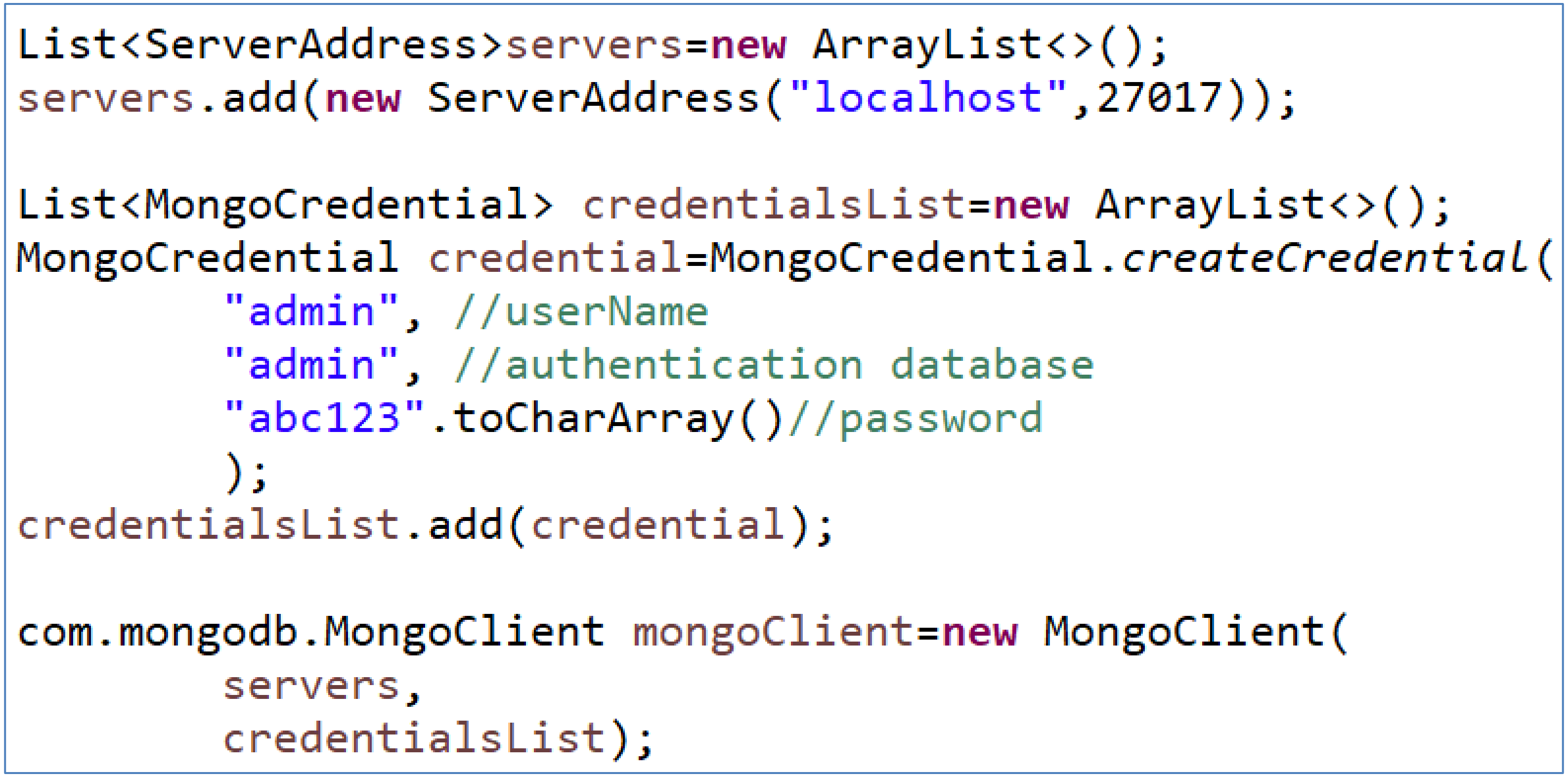
Connect MongoDB – sync driver

- Without authentication



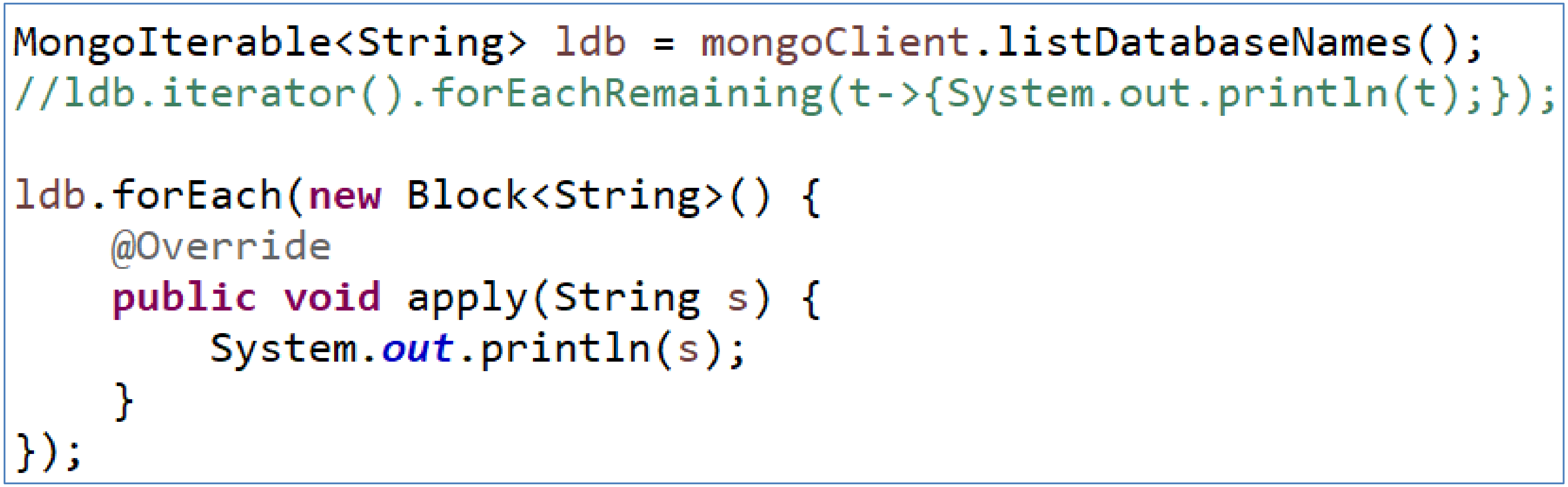
```java
com.mongodb.MongoClient cl=new MongoClient("localhost",27017);
```

- Authentication enable



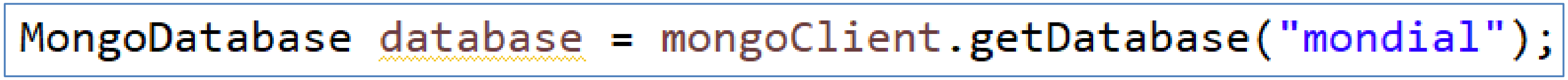
```java
List<ServerAddress>servers=new ArrayList<>();
servers.add(new ServerAddress("localhost",27017));

List<MongoCredential> credentialsList=new ArrayList<>();
MongoCredential credential=MongoCredential.createCredential(
        "admin", //userName
        "admin", //authentication database
        "abc123".toCharArray()//password
        );
credentialsList.add(credential);

com.mongodb.MongoClient mongoClient=new MongoClient(
        servers,
        credentialsList);
```

Get all databases



```
MongoIterable<String> ldb = mongoClient.listDatabaseNames();
//ldb.iterator().forEachRemaining(t->{System.out.println(t);});

ldb.forEach(new Block<String>() {
    @Override
    public void apply(String s) {
        System.out.println(s);
    }
});
```

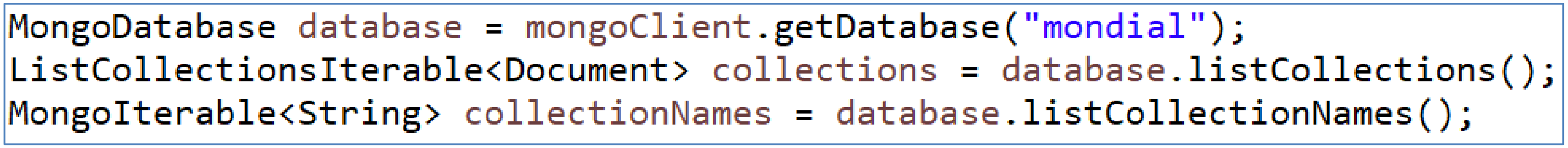
- Get specific database



```
MongoDatabase database = mongoClient.getDatabase("mondial");
```

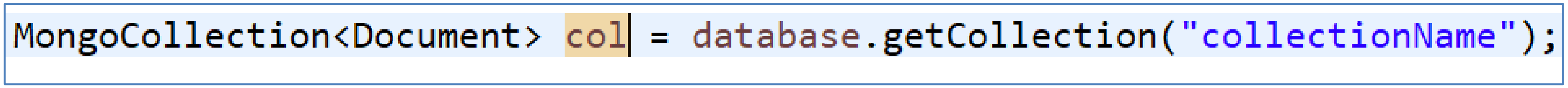
Get collections

- Get all collections



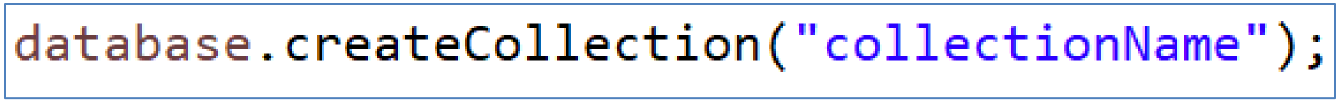
```
MongoDatabase database = mongoClient.getDatabase("mondial");
ListCollectionsIterable<Document> collections = database.listCollections();
MongoIterable<String> collectionNames = database.listCollectionNames();
```

- Get specific collection



```
MongoCollection<Document> col = database.getCollection("collectionName");
```

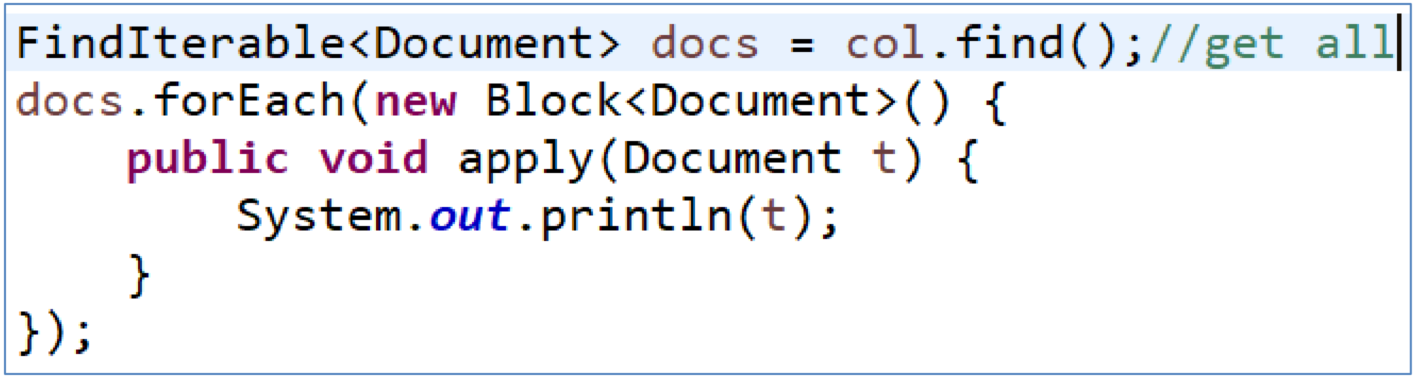
- Create a collection



```
database.createCollection("collectionName");
```

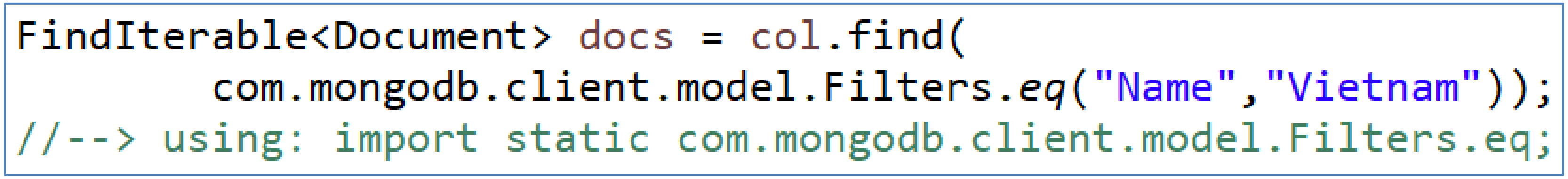
Query

- Get all records



```
FindIterable<Document> docs = col.find();//get all
docs.forEach(new Block<Document>() {
    public void apply(Document t) {
        System.out.println(t);
    }
});
```

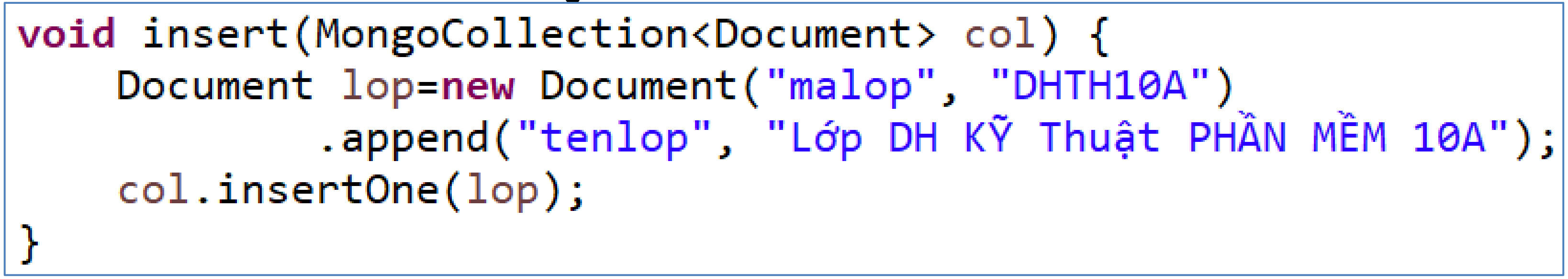
- Filter criteria



```
FindIterable<Document> docs = col.find(
        com.mongodb.client.model.Filters.eq("Name","Vietnam"));
//--> using: import static com.mongodb.client.model.Filters.eq;
```

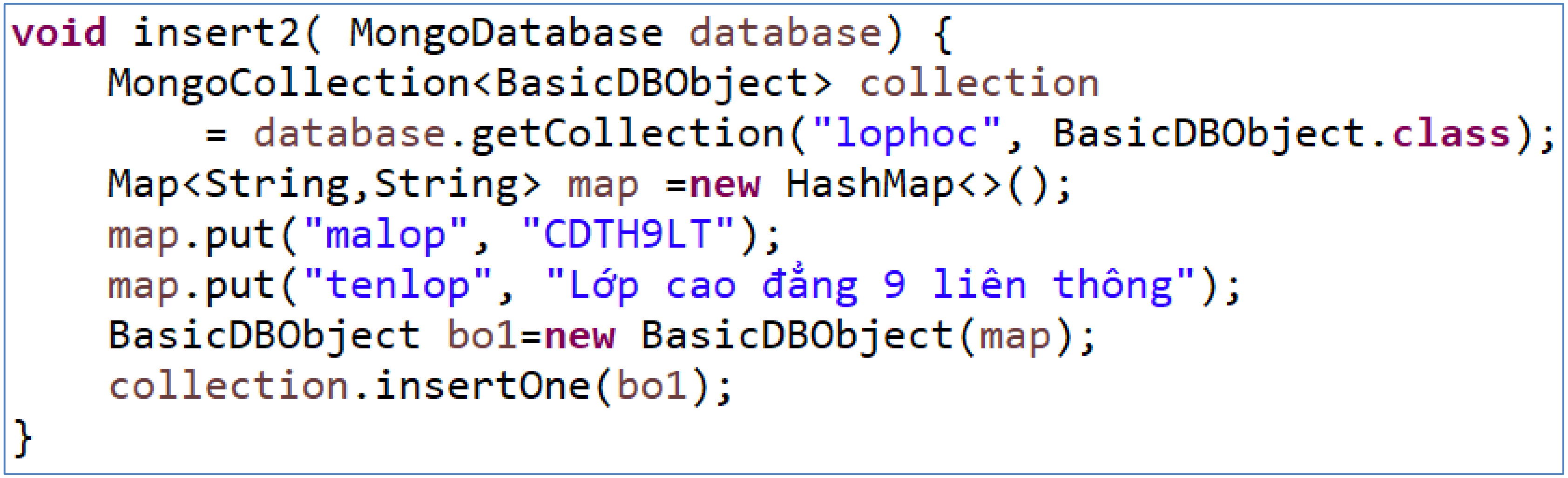
Insert

- Insert a Document object



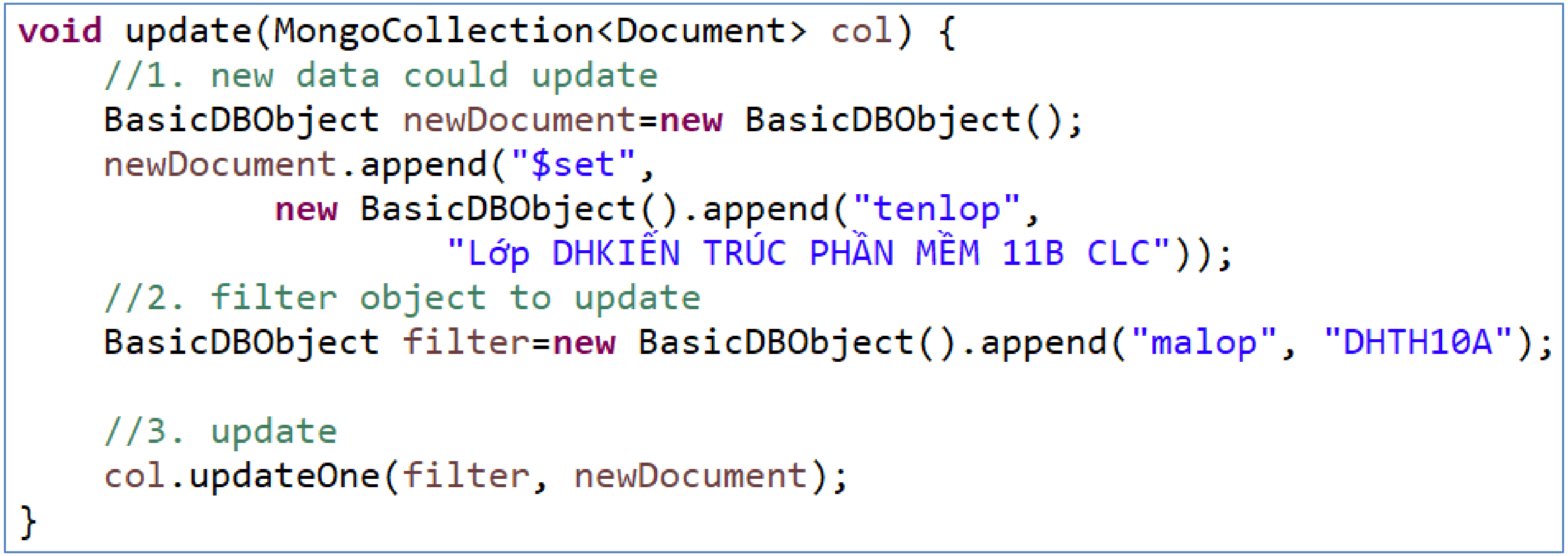
```
void insert(MongoCollection<Document> col) {
    Document lop=new Document("malop", "DHTH10A")
            .append("tenlop", "Lớp DH KỸ Thuật PHẦN MỀM 10A");
    col.insertOne(lop);
}
```

- Insert a BasicDBObject object

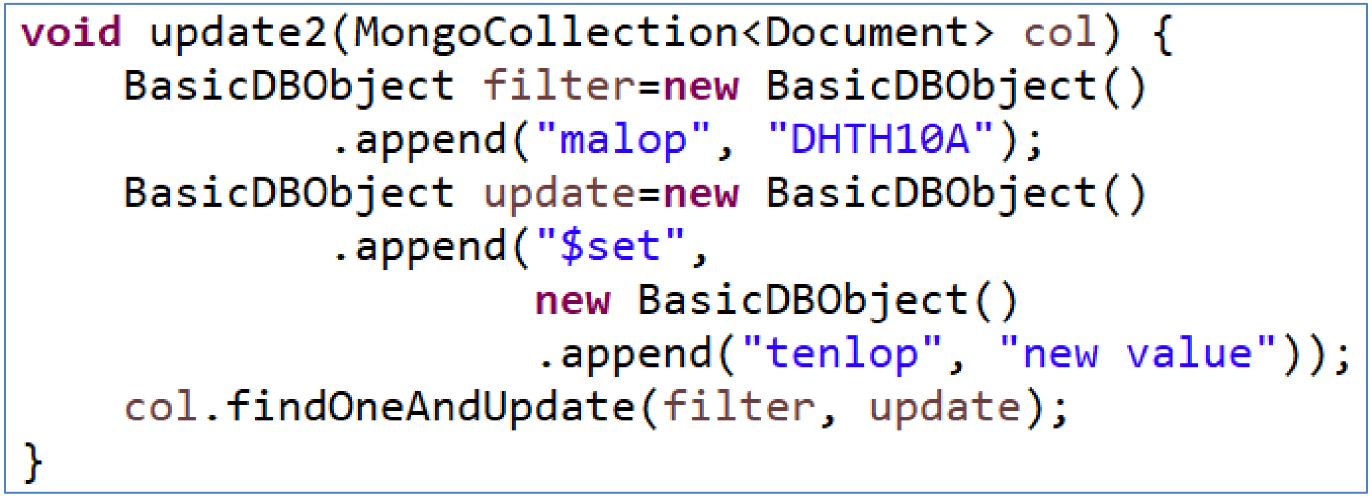


```
void insert2( MongoDatabase database) {
    MongoCollection<BasicDBObject> collection
        = database.getCollection("lophoc", BasicDBObject.class);
    Map<String,String> map =new HashMap<>();
    map.put("malop", "CDTH9LT");
    map.put("tenlop", "Lớp cao đẳng 9 liên thông");
    BasicDBObject bo1=new BasicDBObject(map);
    collection.insertOne(bo1);
}
```

Update

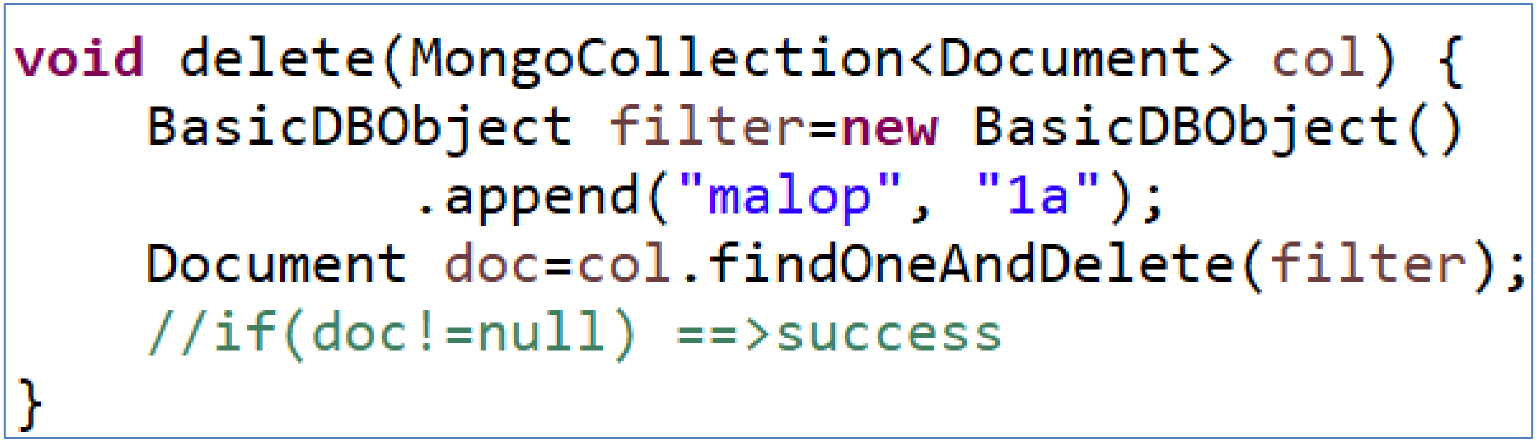


```java
void update(MongoCollection<Document> col) {
    //1. new data could update
    BasicDBObject newDocument=new BasicDBObject();
    newDocument.append("$set",
            new BasicDBObject().append("tenlop",
                    "Lớp DHKIẾN TRÚC PHẦN MỀM 11B CLC"));
    //2. filter object to update
    BasicDBObject filter=new BasicDBObject().append("malop", "DHTH10A");

    //3. update
    col.updateOne(filter, newDocument);
}
```



```java
void update2(MongoCollection<Document> col) {
    BasicDBObject filter=new BasicDBObject()
            .append("malop", "DHTH10A");
    BasicDBObject update=new BasicDBObject()
            .append("$set",
                    new BasicDBObject()
                    .append("tenlop", "new value"));
    col.findOneAndUpdate(filter, update);
}
```

Delete



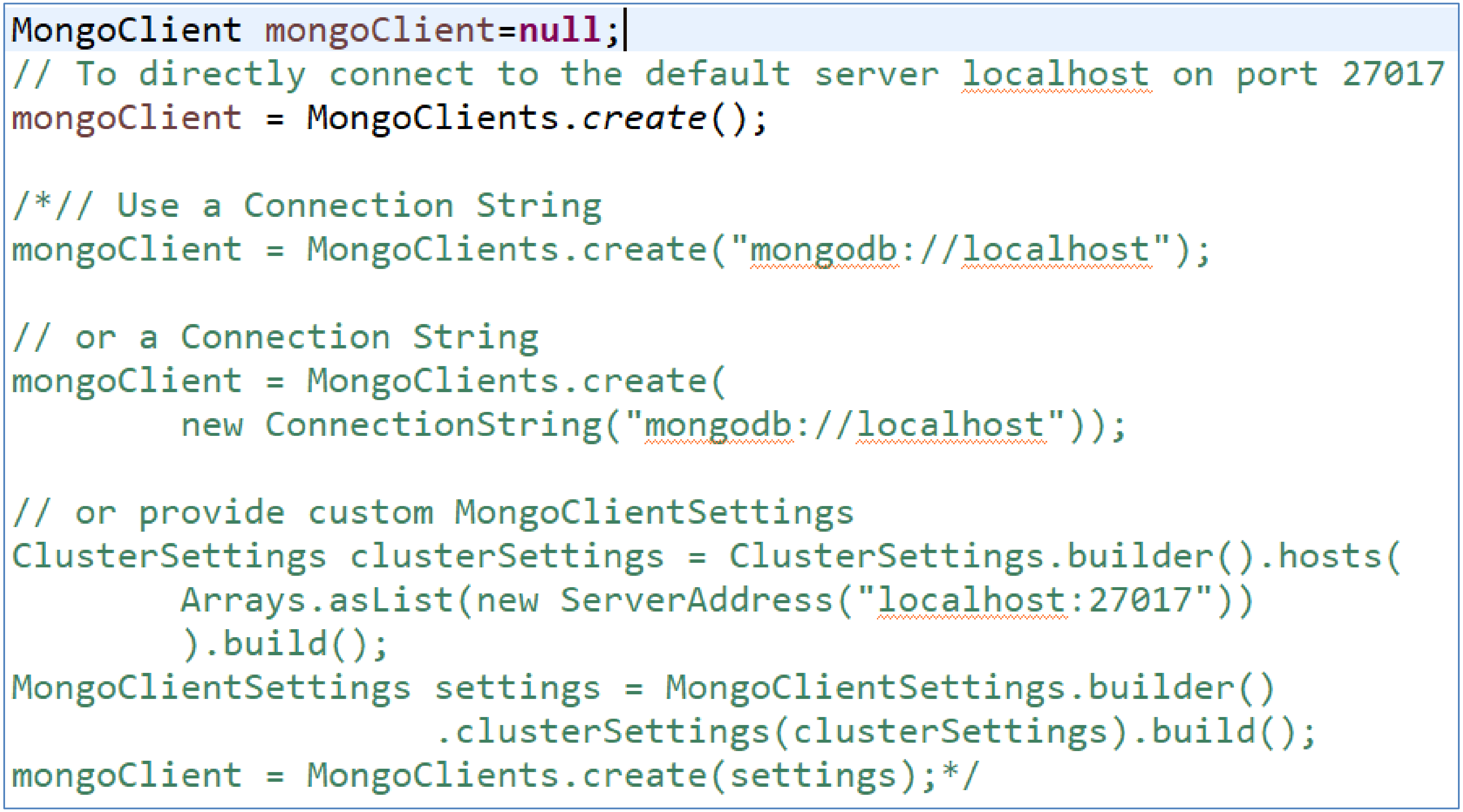
```
void delete(MongoCollection<Document> col) {
    BasicDBObject filter=new BasicDBObject()
            .append("malop", "1a");
    Document doc=col.findOneAndDelete(filter);
    //if(doc!=null) ==>success
}
```

MongoDB – Java accessing asynchronously

- Latency
- Network traffic
- …

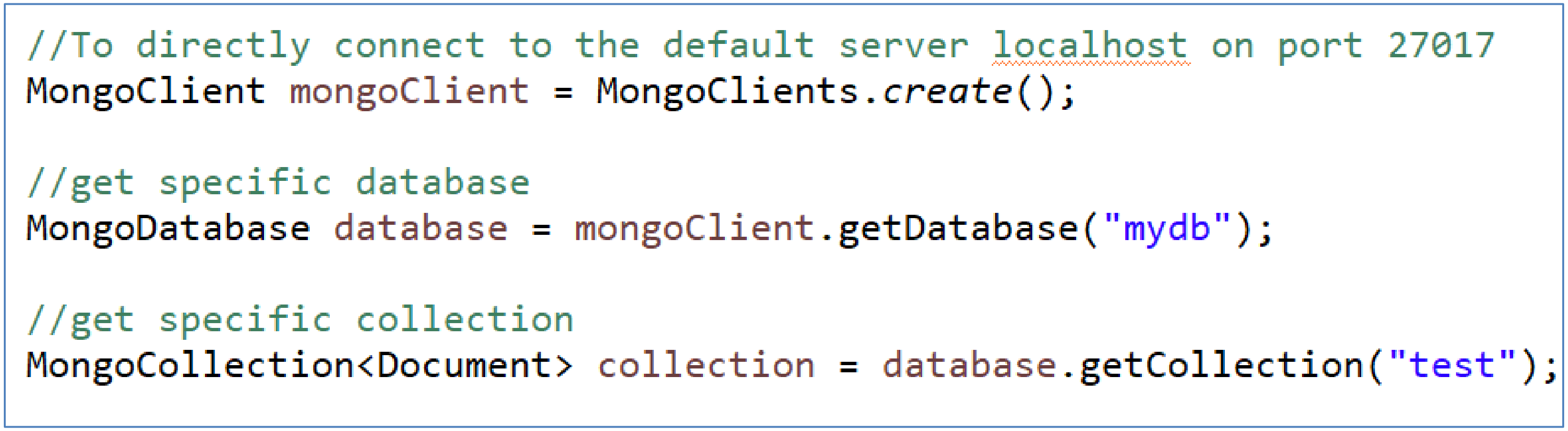
?

Connect MongoDB - Async driver



```
MongoClient mongoClient=null;
// To directly connect to the default server localhost on port 27017
mongoClient = MongoClients.create();

/*// Use a Connection String
mongoClient = MongoClients.create("mongodb://localhost");

// or a Connection String
mongoClient = MongoClients.create(
        new ConnectionString("mongodb://localhost"));

// or provide custom MongoClientSettings
ClusterSettings clusterSettings = ClusterSettings.builder().hosts(
        Arrays.asList(new ServerAddress("localhost:27017"))
        ).build();
MongoClientSettings settings = MongoClientSettings.builder()
                    .clusterSettings(clusterSettings).build();
mongoClient = MongoClients.create(settings);*/
```

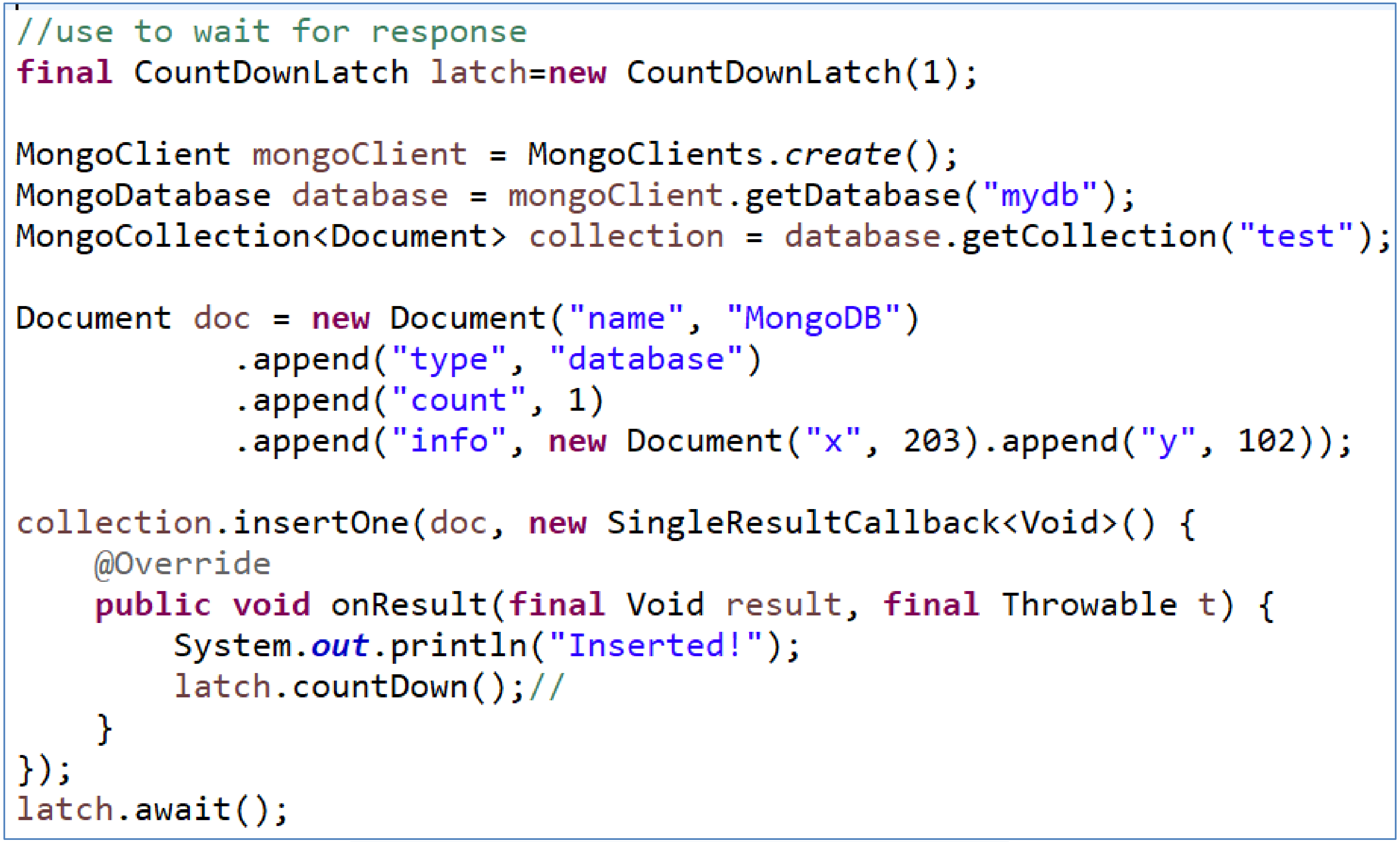
Get Database & collection



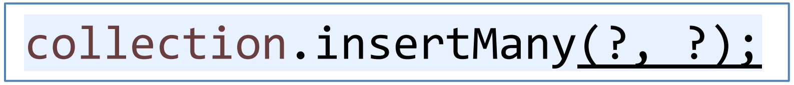
```
//To directly connect to the default server localhost on port 27017
MongoClient mongoClient = MongoClients.create();

//get specific database
MongoDatabase database = mongoClient.getDatabase("mydb");

//get specific collection
MongoCollection<Document> collection = database.getCollection("test");
```

- How about the latency of network tracfic?

Insert

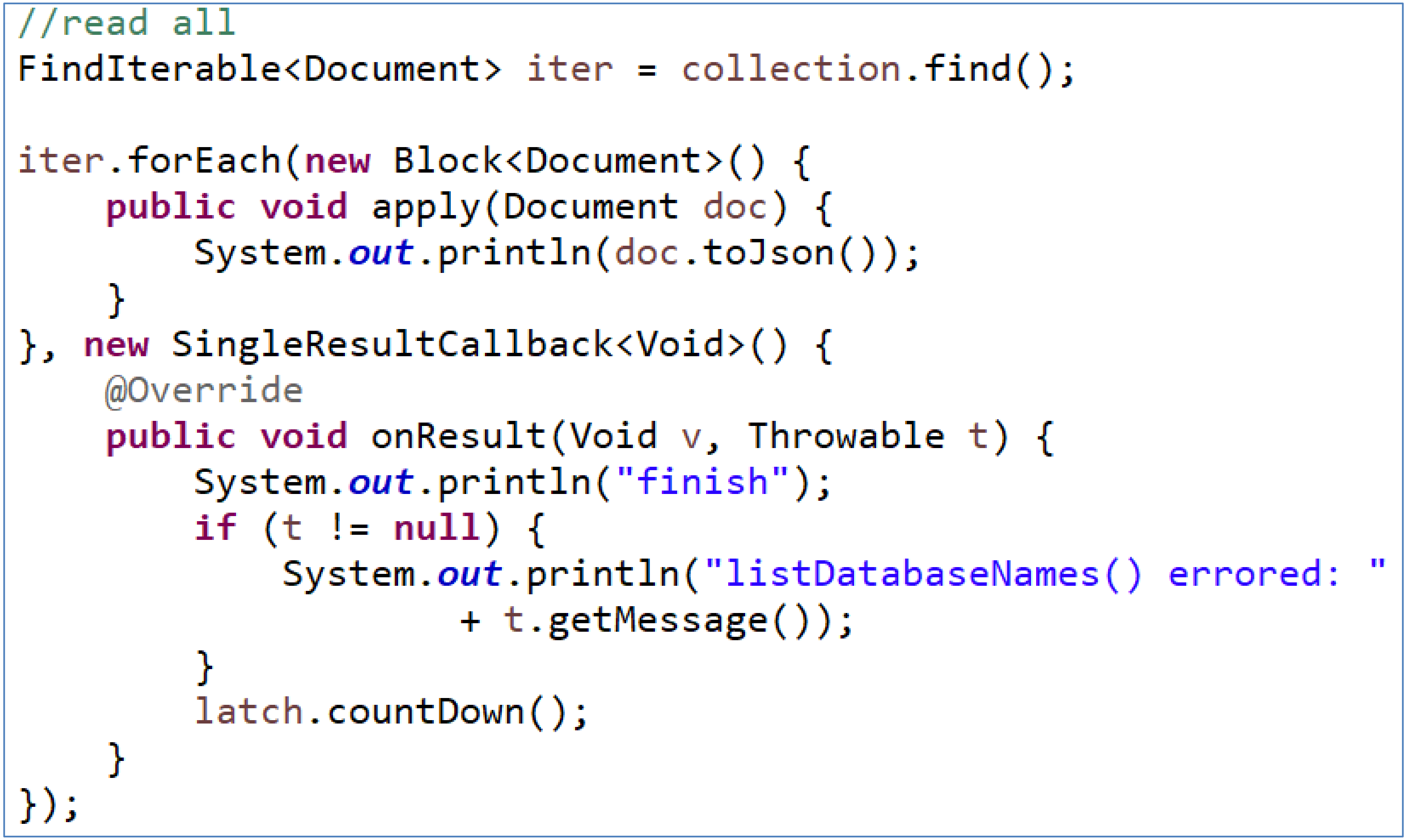


```
//use to wait for response
final CountDownLatch latch=new CountDownLatch(1);

MongoClient mongoClient = MongoClients.create();
MongoDatabase database = mongoClient.getDatabase("mydb");
MongoCollection<Document> collection = database.getCollection("test");

Document doc = new Document("name", "MongoDB")
           .append("type", "database")
           .append("count", 1)
           .append("info", new Document("x", 203).append("y", 102));

collection.insertOne(doc, new SingleResultCallback<Void>() {
    @Override
    public void onResult(final Void result, final Throwable t) {
        System.out.println("Inserted!");
        latch.countDown();//
    }
});
latch.await();
```



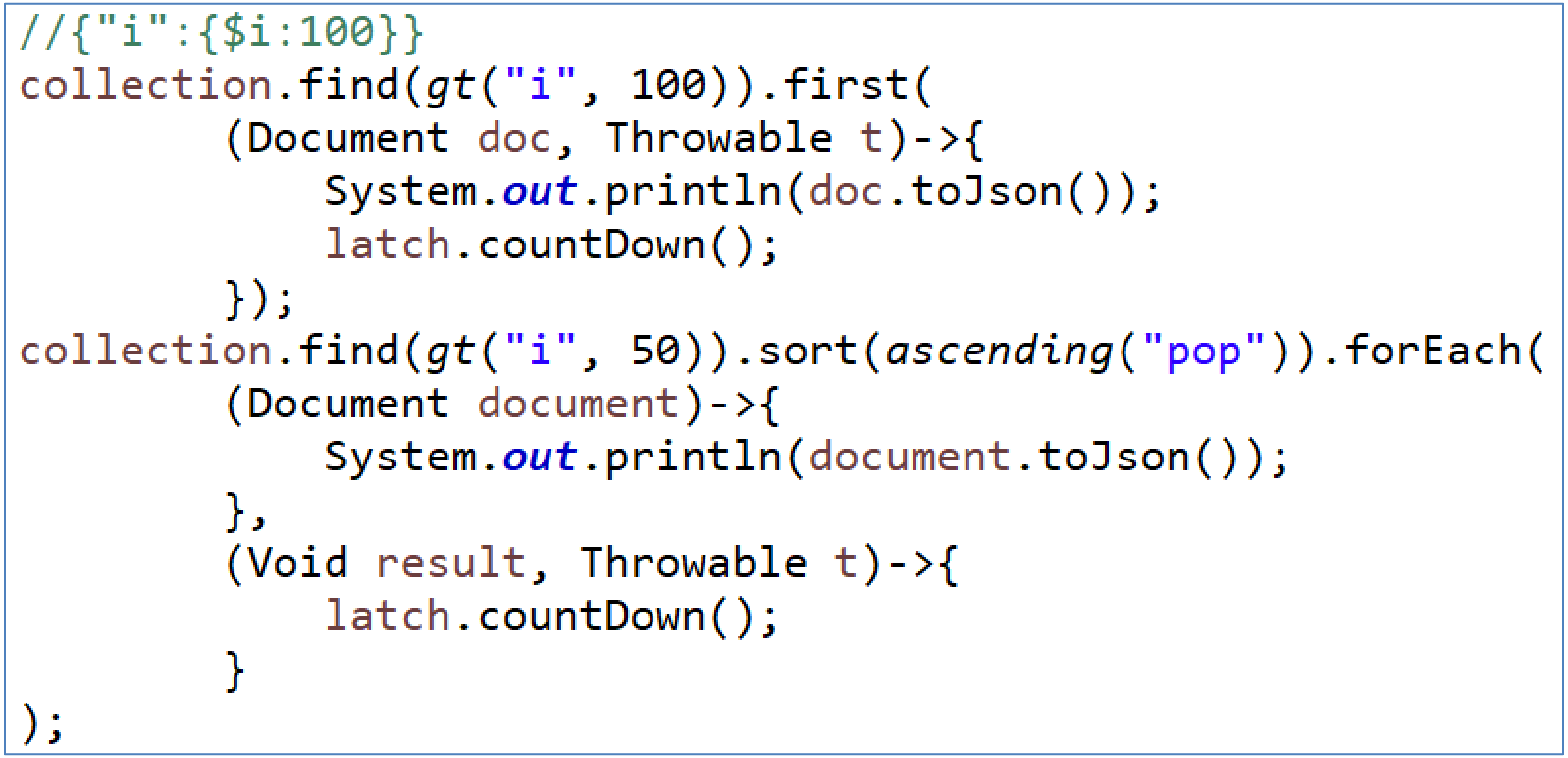
```
collection.insertMany(?, ?);
```

Query



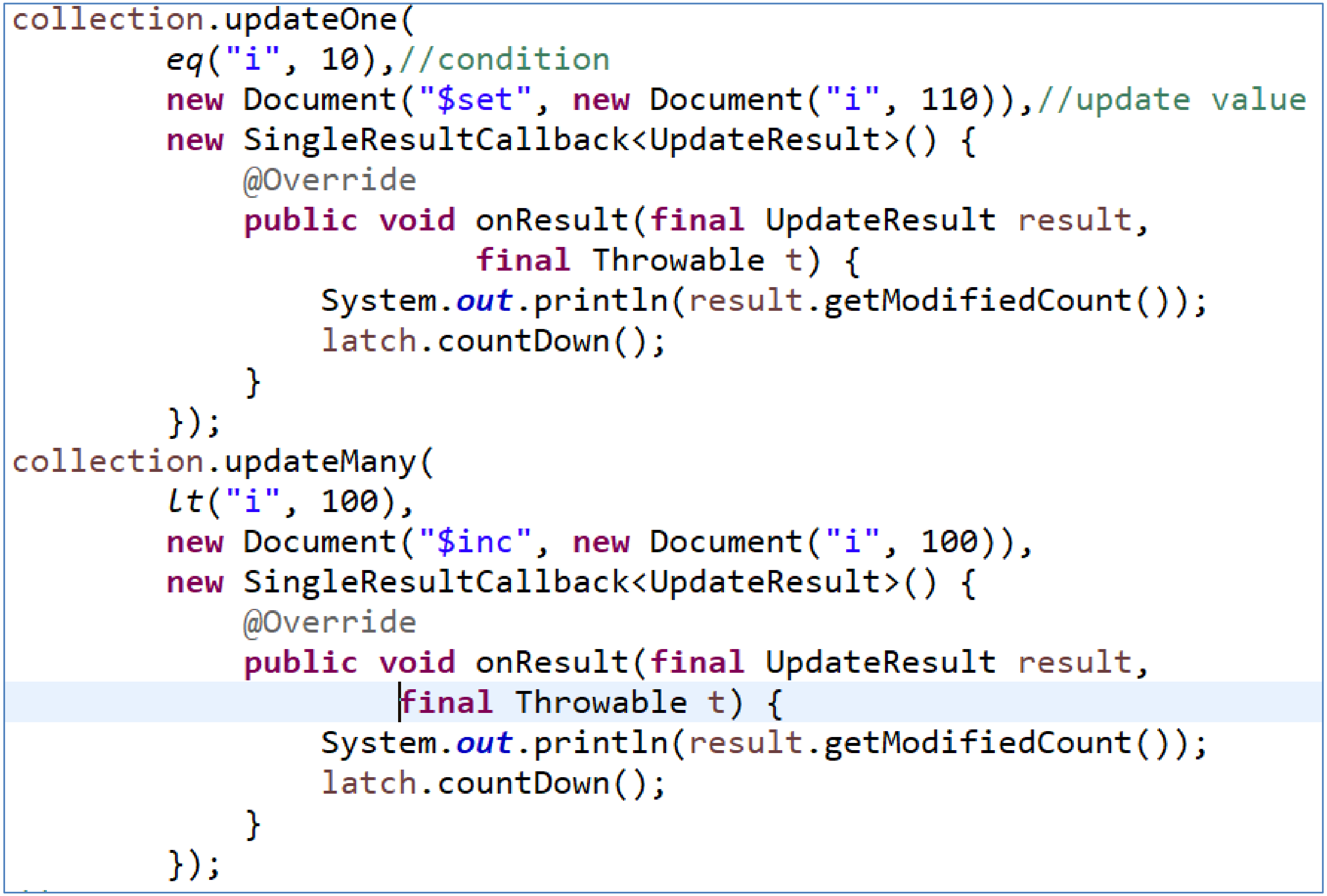
```java
//read all
FindIterable<Document> iter = collection.find();

iter.forEach(new Block<Document>() {
    public void apply(Document doc) {
        System.out.println(doc.toJson());
    }
}, new SingleResultCallback<Void>() {
    @Override
    public void onResult(Void v, Throwable t) {
        System.out.println("finish");
        if (t != null) {
            System.out.println("listDatabaseNames() errored: "
                    + t.getMessage());
        }
        latch.countDown();
    }
});
```

Query with criteria



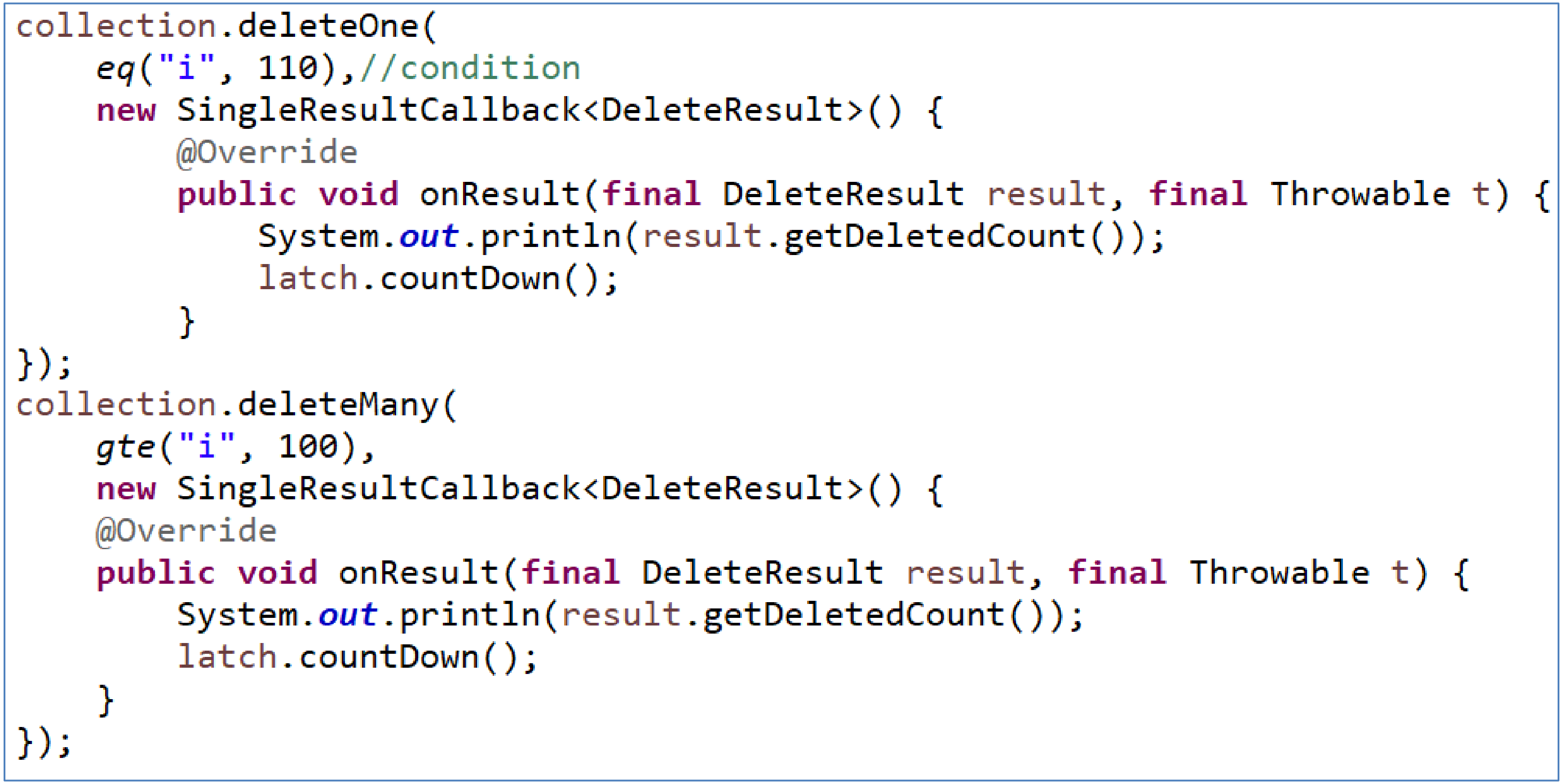
```
//{"i":{$i:100}}
collection.find(gt("i", 100)).first(
        (Document doc, Throwable t)->{
            System.out.println(doc.toJson());
            latch.countDown();
        });
collection.find(gt("i", 50)).sort(ascending("pop")).forEach(
        (Document document)->{
            System.out.println(document.toJson());
        },
        (Void result, Throwable t)->{
            latch.countDown();
        }
);
```

Update



```
collection.updateOne(
        eq("i", 10),//condition
        new Document("$set", new Document("i", 110)),//update value
        new SingleResultCallback<UpdateResult>() {
            @Override
            public void onResult(final UpdateResult result,
                        final Throwable t) {
                System.out.println(result.getModifiedCount());
                latch.countDown();
            }
        });
collection.updateMany(
        lt("i", 100),
        new Document("$inc", new Document("i", 100)),
        new SingleResultCallback<UpdateResult>() {
            @Override
            public void onResult(final UpdateResult result,
                    final Throwable t) {
                System.out.println(result.getModifiedCount());
                latch.countDown();
            }
        });
```

Delete



```
collection.deleteOne(
    eq("i", 110),//condition
    new SingleResultCallback<DeleteResult>() {
        @Override
        public void onResult(final DeleteResult result, final Throwable t) {
            System.out.println(result.getDeletedCount());
            latch.countDown();
        }
});
collection.deleteMany(
    gte("i", 100),
    new SingleResultCallback<DeleteResult>() {
    @Override
    public void onResult(final DeleteResult result, final Throwable t) {
        System.out.println(result.getDeletedCount());
        latch.countDown();
    }
});
```

Summary

Big Data Overview

NoSQL introduction

MongoDB introduction

MongoDB – Java Programming