

# MongoDB Introduction

Ali Jaafar  
SMB214

ali.jaafar86@gmail.com  
01-01-2016



# MongoDB Introduction

Ali Jaafar  
SMB214

ali.jaafar86@gmail.com  
01-01-2016

## Agenda

- What is MongoDB
- why MongoDB
- MongoDB Data Model
- MongoDB Deployment
- MongoDB Query language
- Built-In MapReduce
- Aggregation
- Demo

## Data Model

MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays and other types of documents. MongoDB stores all documents in collections. A collection is a group of related documents that have same and same of documents.

```
{ "name": "John", "age": 30, "gender": "Male", "address": { "street": "123 Main St", "city": "New York", "state": "NY", "zip": "10001" } }
```

## What is MongoDB

MongoDB (from "humongous") is a scalable, high-performance, open source, document-oriented database. MongoDB is classified as NoSQL Database.



## Why MongoDB

MongoDB focuses on four main things:

- Flexibility
- Power
- Speed
- Easy to Use

## MongoDB Query language

```
{ "name": "John", "age": 30, "gender": "Male", "address": { "street": "123 Main St", "city": "New York", "state": "NY", "zip": "10001" } }
```

## MongoDB Deployment

MongoDB runs on many platforms and supports both 32-bit and 64-bit architectures.



## Built-In MapReduce

MapReduce is a programming model for processing large volumes of data in parallel on a cluster of commodity hardware. It is a key feature of MongoDB that allows you to perform complex data processing tasks without the need for external tools like Hadoop.

## Aggregation

Aggregations are operations that process documents and return aggregated results. They are used to perform complex data processing tasks such as grouping, filtering, and calculating statistics. Aggregations are performed on a collection of documents and return a single document as the result.

## Real Example

```
{ "name": "John", "age": 30, "gender": "Male", "address": { "street": "123 Main St", "city": "New York", "state": "NY", "zip": "10001" } }
```

## Real Example

```
{ "name": "John", "age": 30, "gender": "Male", "address": { "street": "123 Main St", "city": "New York", "state": "NY", "zip": "10001" } }
```

## Code

```
use mydb;
insert('users', {name: 'John', age: 30, gender: 'Male', address: {street: '123 Main St', city: 'New York', state: 'NY', zip: '10001'}});
```

Thanks!

# Agenda

- *What is MongoDB*
- *why MongoDB*
- *MongoDB Data Model*
- *MongoDB Deployment*
- *MongoDB Query language*
- *Built-In MapReduce*
- *Aggregation*
- *Demo*

# What is MongoDB

MongoDB (from "humongous") is a scalable, high-performance, open source, document-oriented database

MongoDB is classified as NoSQL Database



# Why MongoDB

MongoDB focuses on four main things:

- Flexibility
- Power
- Speed
- Easy of Use

## *Flexibility*

- Data stored in JSON documents (serialized to BSON)
- Schema-less
- Maps to native programming languages
- ERDs are not governing the design (like RDBMS)

## *Power*

- Supports secondary indexes
- Dynamic Queries
- Sorting
- Rich Updates
- Easy Aggregations
- "Upserts" - update if document exists, insert if it doesn't

## *Speed/Scaling*

- Related data kept together in documents
- No need for joining various tables
- Auto-sharding for scaling clusters linearly
- Can increase capacity with "No Downtime"

## *Ease of Use*

- Very easy to install, configure, maintain, and use
- Very few configuration options
- Works right out of the box
- No need for fine-tuning obscure database configurations

## *Flexibility*

- Data stored in JSON documents (serialized to BSON)
- Schema-less
- Maps to native programming languages
- ERDs are not governing the design (like RDBMS)

## *Power*

- Supports secondary indexes
- Dynamic Queries
- Sorting
- Rich Updates
- Easy Aggregations
- “Upserts” –update if document exists, insert if it doesn’t

## *Speed/Scaling*

- Related data kept together in documents
- No need for joining various tables
- Auto-sharding allow for scaling clusters linearly
- Can increase capacity with “No Downtime”



## ***Ease of Use***

- Very easy to install, configure, maintain, and use
- Very few configuration options
- Works right out of the box
- No need for fine-tuning obscure database configurations

# Data Model

MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information.

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

MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value  
← field: value  
← field: value  
← field: value

```
{  
  name: "al",  
  age: 18,  
  status: "D",  
  groups: [ "politics", "news" ]  
}
```

Collection

# Data Model

A database holds a set of collections (Mongo DB ~= SQL DB)

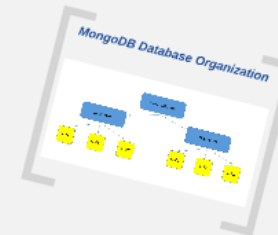
A collection holds a set of documents (Mongo Collection ~= SQL Table)

A document is a set of fields (Mongo Documents ~= SQL Record)

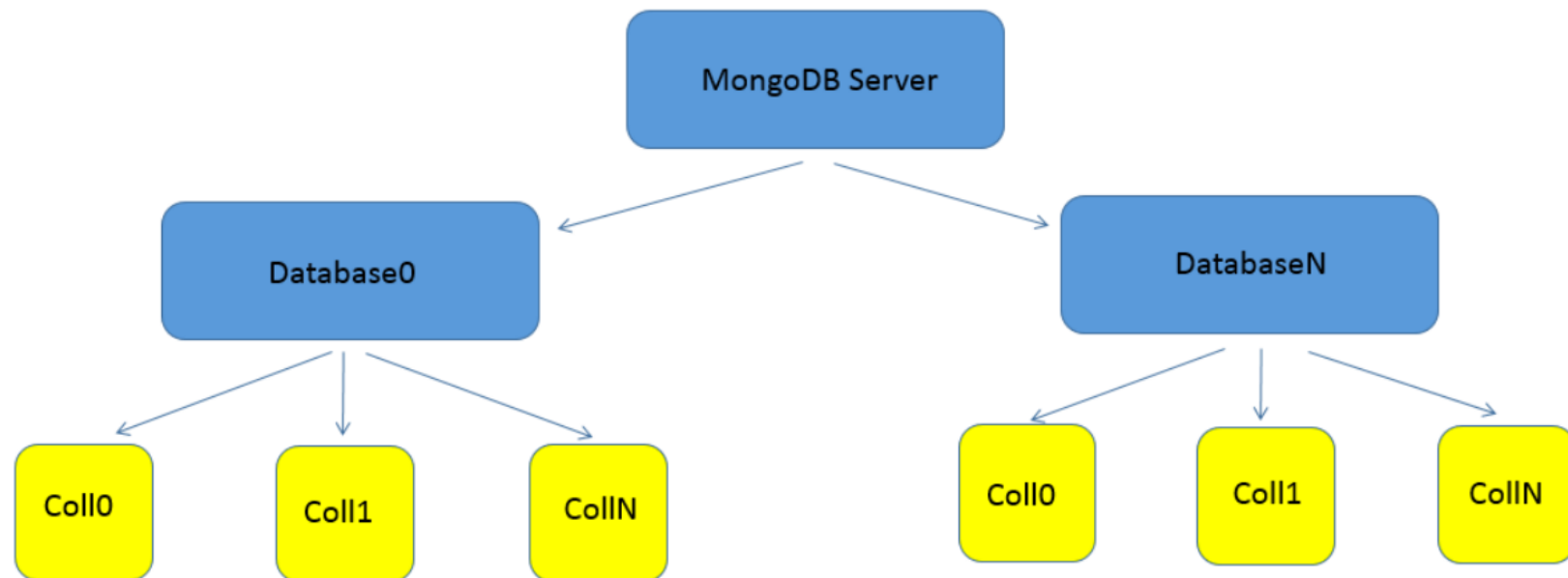
A field is a key-value pair (Mongo Field ~= SQL Attribute)

A key is a name , a value can be :

- a data type like string,integer,float,binary...
- a document
- an array of Values



# *MongoDB Database Organization*



# Example Tabular Structure

**Person**

ID	FIRST_NAME	LAST_NAME
1	Ali	Jaafar

 **Account**

ID	ACCOUNT_TYPE	ACCOUNT_BALANCE	CURRENCY	(FK:Person)
1	Investment	80000.00	USD	1
2	Savings	70400.00	LBP	1

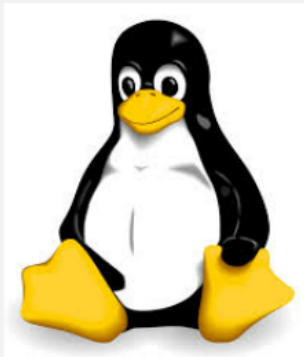
# *Example Document Oriented Structure*

Person Collection (Person is the root entity)

```
[{
  "ID": 1,
  "first_name": "Ali",
  "last_name": "Jaafar",
  "accounts": [
    {
      "id": 1,
      "account_type": "Savings",
      "account_balance": "70400.00",
      "currency": "USD"
    },
    {
      "id": 2,
      "account_type": "Checking",
      "account_balance": "80000.00",
      "currency": "LBP"
    }
  ]
}]
```

# MongoDB Deployment

MongoDB runs on most platforms and supports both 32-bit and 64-bit architectures.



## *Deployment on windows*

- Download the Zip File 32-Bit or 64-Bit (recommended)
- Unzip the Download
- Create a Data Directory

```
C:\> mkdir \data  
C:\> mkdir \data\db
```

## *Run & Connect to the Server*

```
C:\> cd \my_mongo_dir\bin  
C:\my_mongo_dir\bin> mongod
```





# MongoDB Query language

MySQL term	Mongo term/concept
database	database
table	collection
index	index
row	BSON document
column	BSON field
join	embedding and linking
primary key	_id field
group by	aggregation

# MongoDB Query language

```
CREATE TABLE USERS (a Number, b Number)
```

implicit; can also be done explicitly with

```
db.createCollection("mycoll")
```

```
ALTER TABLE users ADD ...
```

implicit

```
INSERT INTO USERS VALUES (3,5)
```

```
db.users.insert({a:3,b:5})
```

```
CREATE INDEX myindexname ON users(name)
```

```
db.users.ensureIndex({name:1})
```

```
CREATE INDEX myindexname ON users(name,ts  
DESC)
```

```
db.users.ensureIndex({name:1,ts:-1})
```

# MongoDB Query language

```
SELECT a,b FROM users
```

```
db.users.find({}, {a:1,b:1})
```

```
SELECT * FROM users
```

```
db.users.find()
```

```
SELECT * FROM users WHERE age=33
```

```
db.users.find({age:33})
```

```
SELECT a,b FROM users WHERE age=33
```

```
db.users.find({age:33}, {a:1,b:1})
```

```
SELECT * FROM users WHERE age=33 ORDER BY  
name
```

```
db.users.find({age:33}).sort({name:1})
```

# MongoDB Query language

```
EXPLAIN SELECT * FROM users WHERE z=3
```

```
db.users.find({z:3}).explain()
```

```
UPDATE users SET a=1 WHERE b='q'
```

```
db.users.update({b:'q'}, {$set:{a:1}},  
false, true)
```

```
UPDATE users SET a=a+2 WHERE b='q'
```

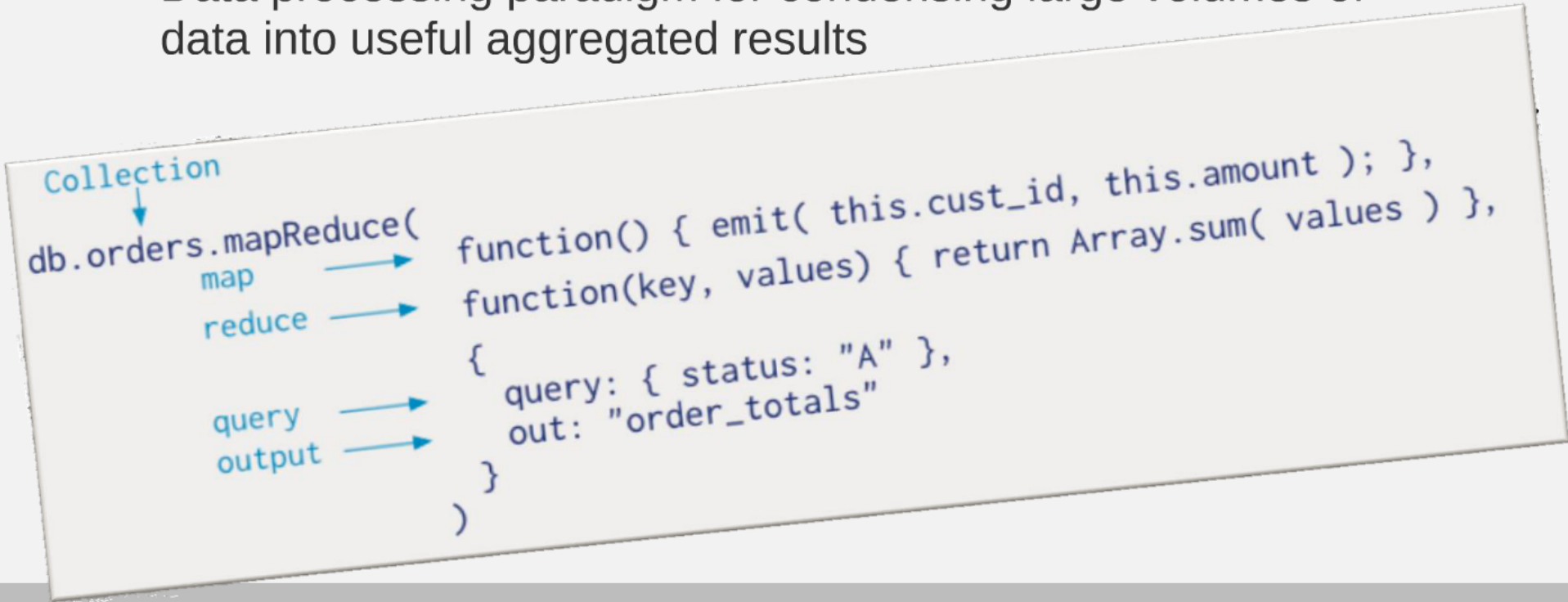
```
db.users.update({b:'q'}, {$inc:{a:2}},  
false, true)
```

```
DELETE FROM users WHERE z="abc"
```

```
db.users.remove({z:'abc'});
```

# Built-In MapReduce

Data processing paradigm for condensing large volumes of data into useful aggregated results



The diagram shows a MongoDB MapReduce command with several annotations. A blue arrow points from the word "Collection" to the `db.orders` part of the command. Another blue arrow points from the word "map" to the `map` parameter. A third blue arrow points from the word "reduce" to the `reduce` parameter. Two blue arrows point from the words "query" and "output" to the `query` and `out` fields of the options object, respectively.

```
db.orders.mapReduce(  
  map  
  reduce  
  {  
    query: { status: "A" },  
    out: "order_totals"  
  }  
)
```

# Built-In MapReduce

Map Reduce is the process of processing the entire database with 2 steps Map and Reduce

Map Phase—processes each document and emits one or more objects for each document

Reduce Phase—combines the output of the map operation

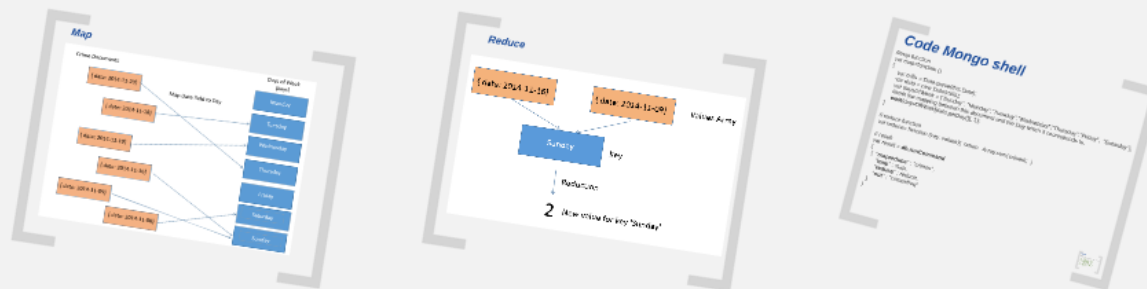
Finalize (optional) —used to make final modifications to the output

# Real Example

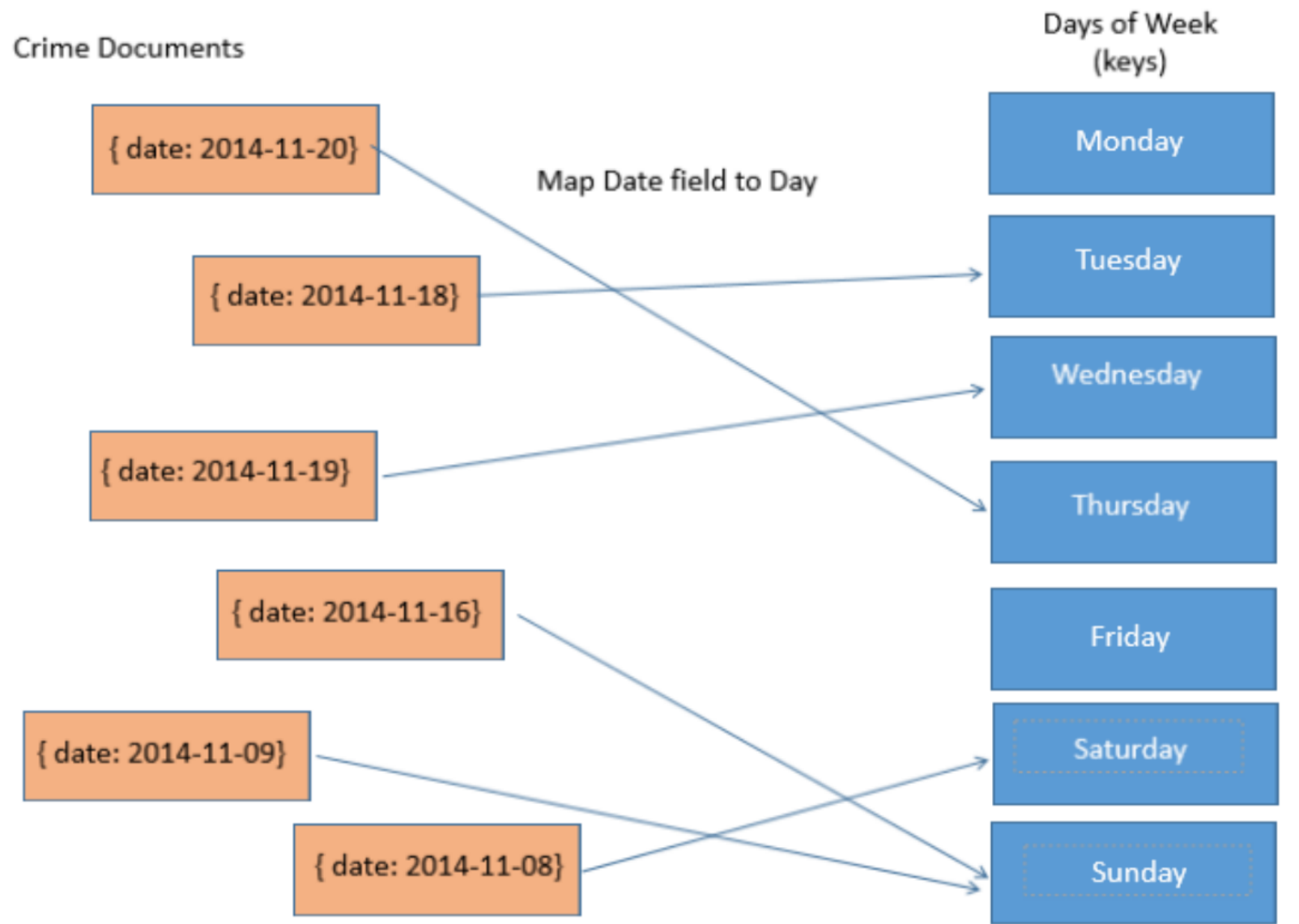
The example data set we will be dealing with is City of Chicago crime police report data from November 2014 to December 2015.

```
{ "_id" : ObjectId("5462725476ecd357dbbc721e"), "ID" : 9844675, "Case Number" : "HX494115", "Date" : "11/03/2014 11:51:00 PM", "Block" : "056XX S MORGAN ST", "IUCR" : 486, "Primary Type" : "BATTERY", "Description" : "DOMESTIC BATTERY SIMPLE", "Location Description" : "ALLEY", "Arrest" : "false", "Domestic" : "true", "Beat" : 712, "District" : 7, "Ward" : 16, "Community Area" : 68, "FBI Code" : "08B", "X Coordinate" : 1170654, "Y Coordinate" : 1867165, "Year" : 2014, "Updated On" : "11/10/2014 12:43:02 PM", "Latitude" : 41.790980835, "Longitude" : -87.649786614, "Location" : "(41.790980835, -87.649786614)" }
```

**What day of the week has the most crime incidents recorded in Chicago?**

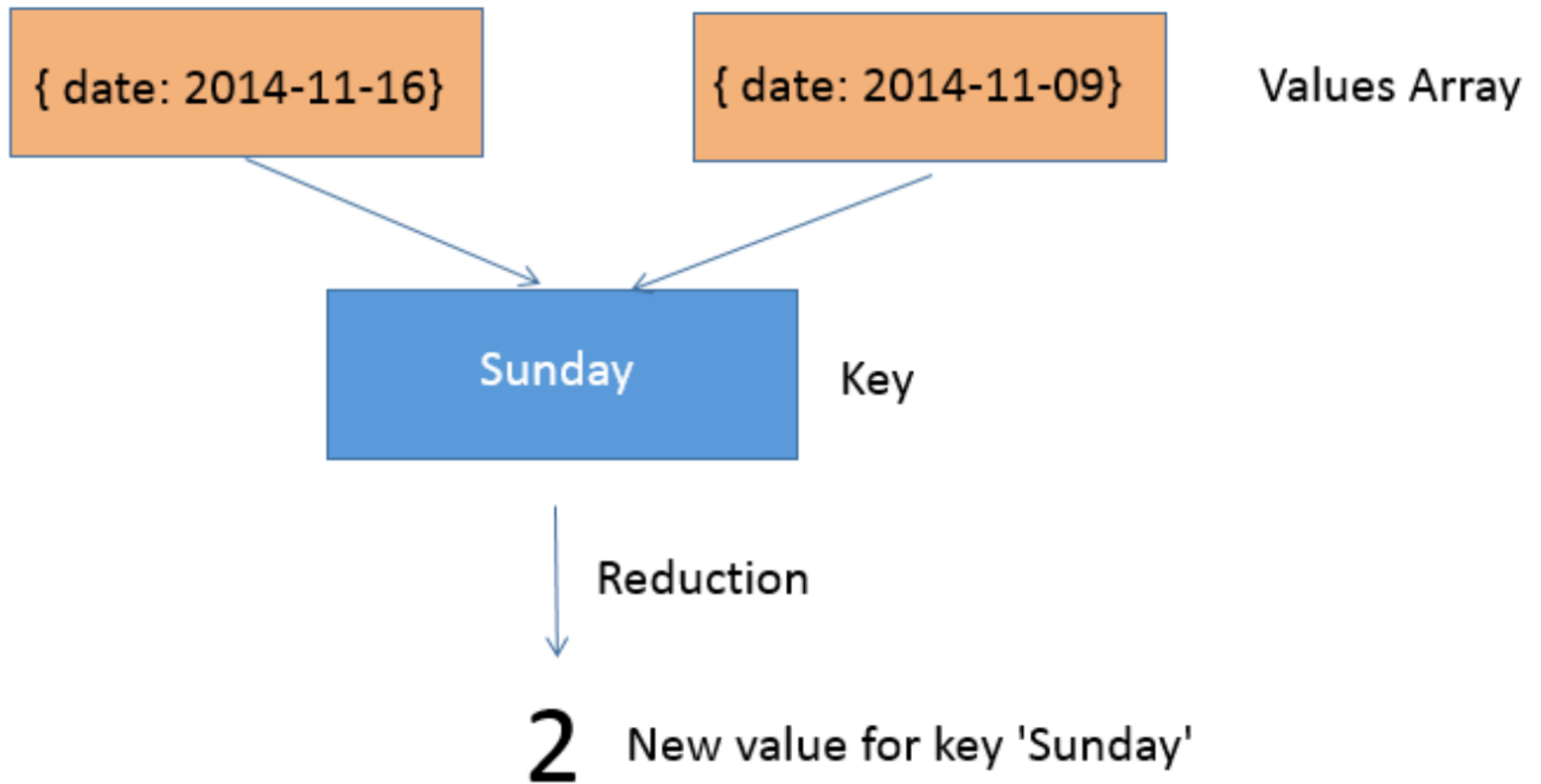


# Map





## Reduce



# Code Mongo shell

```
//map function
var map=function ()
{
  var milis = Date.parse(this.Date);
  var date = new Date(milis);
  var daysOfWeek = ["Sunday", "Monday","Tuesday","Wednesday","Thursday","Friday", "Saturday"];
  //emit the mapping between this document and the Day which it cooresponds to.
  emit(daysOfWeek[date.getDay()], 1);
}

// reduce function
var reduce= function (key, values){ return  Array.sum(values); }

// result
var result = db.runCommand
(
  { "mapreduce" : "crimes",
    "map" : map,
    "reduce" : reduce,
    "out" : "crimesfreq"
  }
)
```

**Results**

Number of crimes based on each day of the week

```
[{"_id": "Sunday", "value": 43181},
{"_id": "Monday", "value": 43271},
{"_id": "Tuesday", "value": 43361},
{"_id": "Wednesday", "value": 43451},
{"_id": "Thursday", "value": 43541},
{"_id": "Friday", "value": 43631},
{"_id": "Saturday", "value": 43721}]
```

# Results

Number of crimes based on each day of the week

```
{ "_id" : "Friday", "value" : 44449 }  
{ "_id" : "Monday", "value" : 43027 }  
{ "_id" : "Saturday", "value" : 44305 }  
{ "_id" : "Sunday", "value" : 42866 }  
{ "_id" : "Thursday", "value" : 41974 }  
{ "_id" : "Tuesday", "value" : 42071 }  
{ "_id" : "Wednesday", "value" : 42569 }
```

# Aggregation

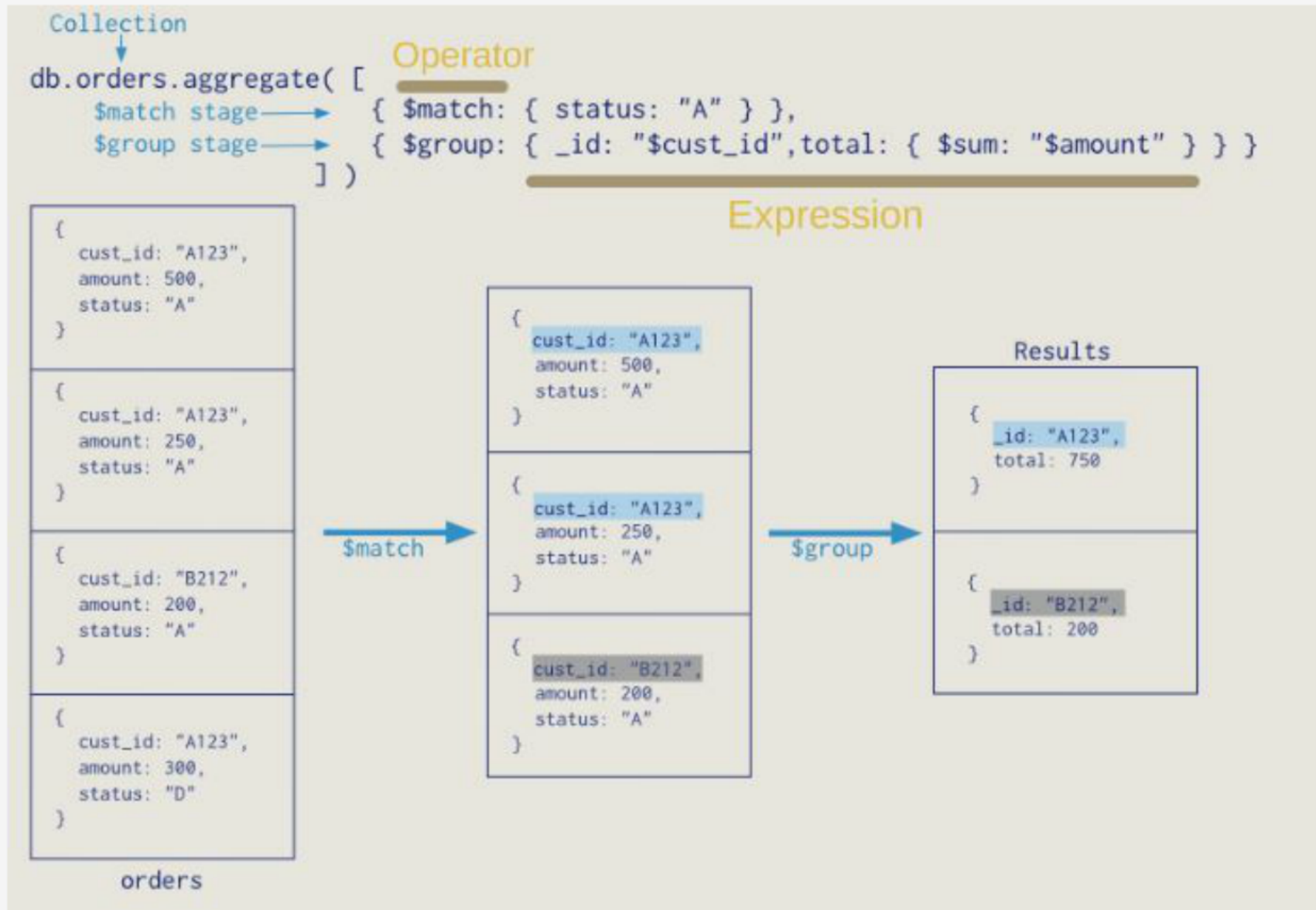
Aggregations are operations that process data records and return computed results.

Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

In sql `count(*)` and with `group by` is an equivalent of mongodb aggregation.

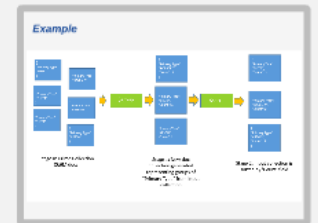


# Aggregation

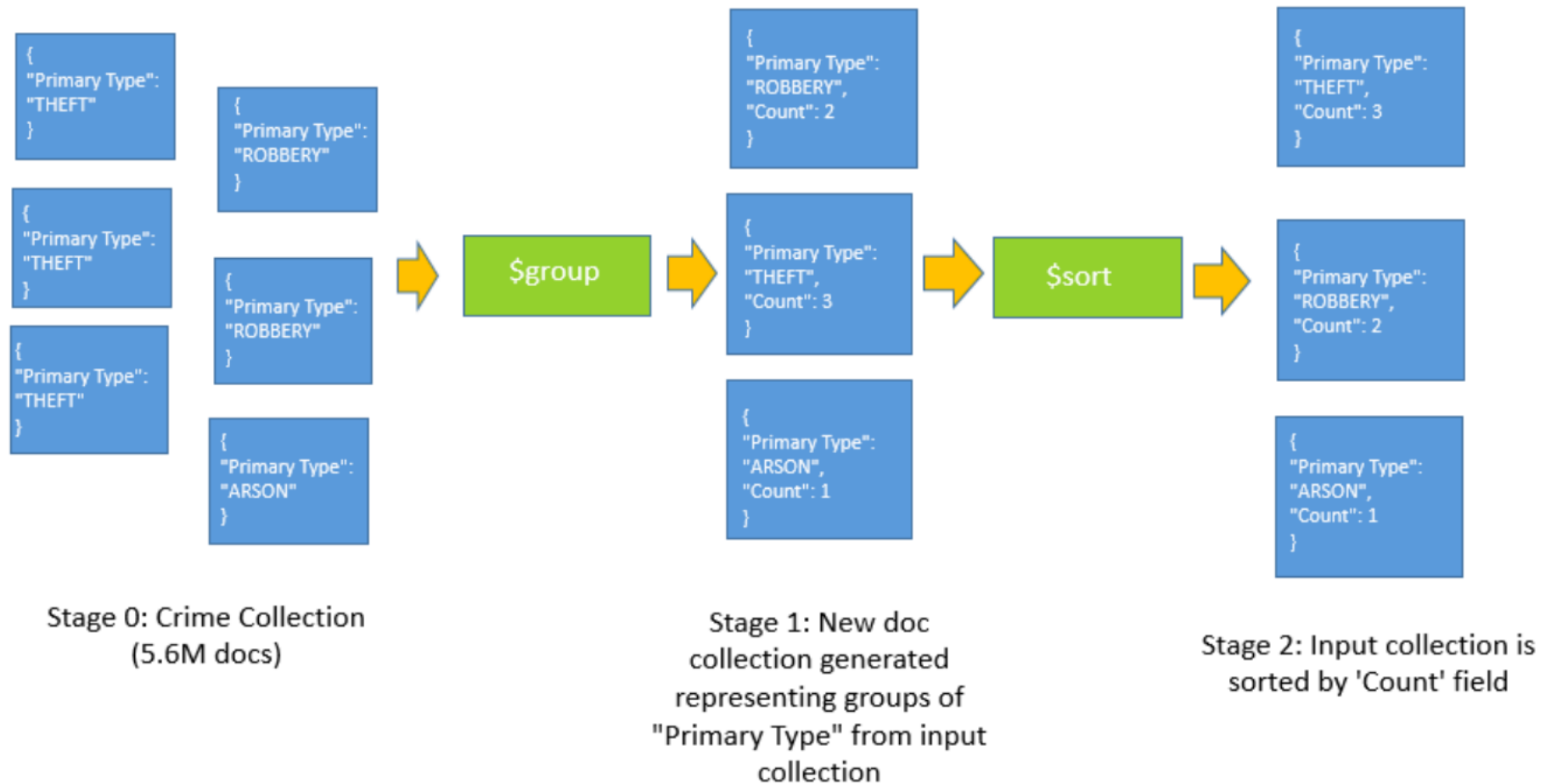


## Real Example

**What is the most common type of crime committed in Chicago between November 2014 and December 2015?**



# Example



## Code

```
db.crimes.aggregate(
    {
        $group : { _id: "$Primary Type" , count:{$sum: 1}  }
    },
    {
        $sort: { count: -1 }
    }
)
```

```

["ST", "THEFT", "count", 6446]
["ST", "BATTERY", "count", 5679]
["ST", "CRIMINAL DAMAGE", "count", 2228]
["ST", "NARCOTICS", "count", 2674]
["ST", "OTHER OFFENSE", "count", 1267]
["ST", "MISDEMT", "count", 1844]
["ST", "RECEIVE/POSSESS", "count", 3659]
["ST", "BRIBE/APP", "count", 1406]
["ST", "MOTOR VEHICLE THEFT", "count", 2246]
["ST", "ROBBERY", "count", 9249]
["ST", "CRIMINAL TRESPASS", "count", 7566]
["ST", "INTA/SEX VIOLENT", "count", 2141]

```



# Results

```
{ "_id" : "THEFT", "count" : 66446 }  
{ "_id" : "BATTERY", "count" : 56372 }  
{ "_id" : "CRIMINAL DAMAGE", "count" : 33281 }  
{ "_id" : "NARCOTICS", "count" : 25799 }  
{ "_id" : "OTHER OFFENSE", "count" : 19937 }  
{ "_id" : "ASSAULT", "count" : 19494 }  
{ "_id" : "DECEPTIVE PRACTICE", "count" : 16359 }  
{ "_id" : "BURGLARY", "count" : 15636 }  
{ "_id" : "MOTOR VEHICLE THEFT", "count" : 11805 }  
{ "_id" : "ROBBERY", "count" : 11408 }  
{ "_id" : "CRIMINAL TRESPASS", "count" : 7566 }  
{ "_id" : "WEAPONS VIOLATION", "count" : 3813 }
```

.....

**DEMO**

***Thanks!***

# MongoDB Introduction

Ali Jaafar  
SMB214

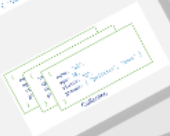
ali.jaafar86@gmail.com  
01-01-2016

## Agenda

- What is MongoDB
- why MongoDB
- MongoDB Data Model
- MongoDB Deployment
- MongoDB Query language
- Built-In MapReduce
- Aggregation
- Demo

## Data Model

MongoDB documents are BSON documents. BSON is a binary representation of JSON with additional type information. In the documents, the value of a field can be any of the BSON data types, including other documents, arrays and other types of documents. MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes.



## What is MongoDB

MongoDB (from "humongous") is a scalable, high-performance, open source, document-oriented database. MongoDB is classified as NoSQL Database.



## Why MongoDB

MongoDB focuses on four main things:

- Flexibility
- Power
- Speed
- Easy to Use

## MongoDB Query language

Field	Value
name	John
age	30
address	123 Main St
city	New York
state	NY
zip	10001
country	USA

## Built-In MapReduce

Aggregations are operations that process documents and return aggregated results. Aggregation operations process documents in a pipeline. The first stage of the aggregation pipeline is the \$match stage, which filters documents based on the criteria specified in the \$match operator. The second stage is the \$group stage, which groups documents by a common field. The final stage is the \$out stage, which writes the results to a collection or to the \$out operator.

## Aggregation

Aggregations are operations that process documents and return aggregated results. Aggregation operations process documents in a pipeline. The first stage of the aggregation pipeline is the \$match stage, which filters documents based on the criteria specified in the \$match operator. The second stage is the \$group stage, which groups documents by a common field. The final stage is the \$out stage, which writes the results to a collection or to the \$out operator.

## MapReduce

MapReduce is a programming model for processing large volumes of data in parallel. It consists of three main stages: Map, Reduce, and Shuffle. The Map stage processes the input data and outputs key-value pairs. The Shuffle stage groups the key-value pairs by key. The Reduce stage processes the grouped data and outputs the final results.

## MapReduce

MapReduce is a programming model for processing large volumes of data in parallel. It consists of three main stages: Map, Reduce, and Shuffle. The Map stage processes the input data and outputs key-value pairs. The Shuffle stage groups the key-value pairs by key. The Reduce stage processes the grouped data and outputs the final results.

Thanks!