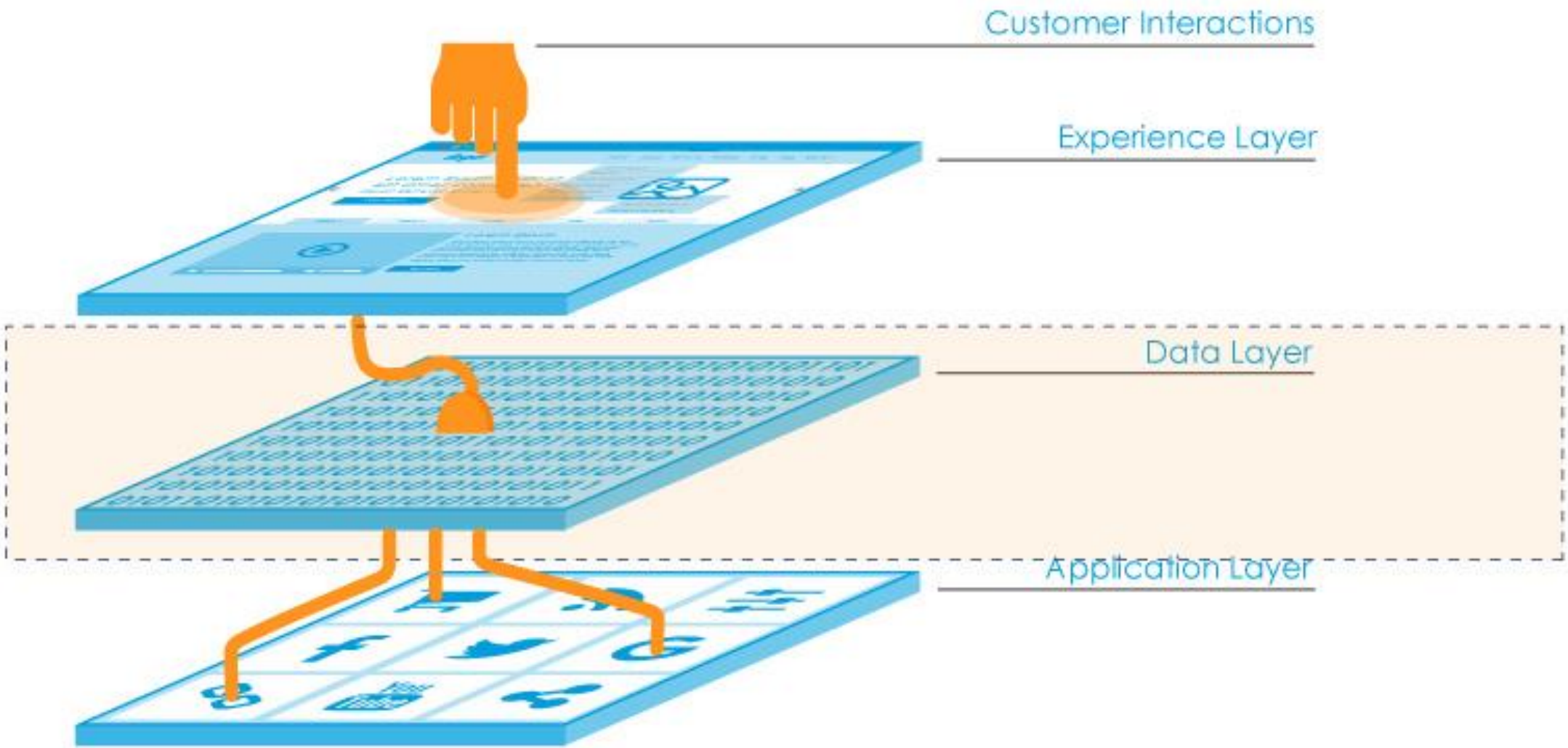

ALSO
BIG
DATA
TOOLS
DISK
TARGET
APPLIED
SENSOR
SHARED
DEFINITION
CURRENT
MAY
MOVING
WITHIN
EXAMPLES
EVERY
RECONSIDER
PRACTITIONERS
CAPTURE
ZETTABYTES
THOUGHT
BUSINESS
INTERNET
DISTRIBUTED
SETS
GENOMICS
DESCRIBING
RADIO-FREQUENCY
COMPLEXITY
MANAGEMENT
TERABYTES
CASE
ABILITY
PETABYTES
INCLUDE
ELAPSED
SYSTEMS
TOLERABLE
FORMS
SIZE
STORAGE
MPP
PARALLEL
SAN
QUALITIES
SINCE
MASSIVELY
GROW
WORKING
OPPORTUNITIES
AMOUNT
GARTNER
CURRENTLY
SOLID
TYPES
DIFFICULTY
SEARCH
RECORDS
NETWORKS
DATABASES
HUNDREDS
DESKTOP
CAPACITY
BURIED
WORLD'S
NOW
BIOLOGICAL
PROCESSING
UBIQUITOUS
ANALYTICS
TENS
COMPLEX
CITATION
INDEXING
LARGER
CONTINUES
SOCIAL
RELATIONAL
ORGANIZATIONS
CONNECTOMICS
USE
SET

NYU

Big Data Storage Layers


Customer Interactions
Experience Layer
Data Layer
Application Layer

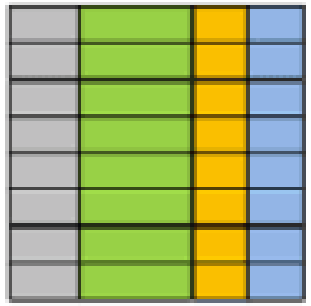
NoSQL

- Not only SQL or NoSQL is a new set of a database that has emerged in the recent as an alternative solution to relational databases
- Not based on the Relational Database Management Systems principles (RDBMS)
- No single product or technology
- Does not require a fixed schema, avoids joins, and is easy to scale

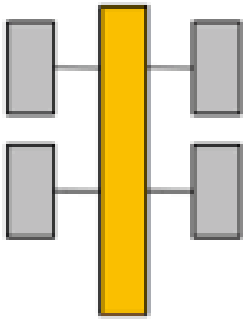
References:
https://www.simplilearn.com/introduction-to-nosql-databases-tutorial-video
https://www.guru99.com/nosql-tutorial.html

SQL Database

Relational



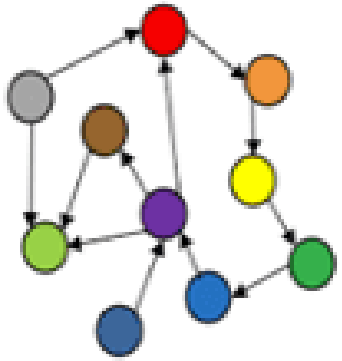
Analytical (OLAP)



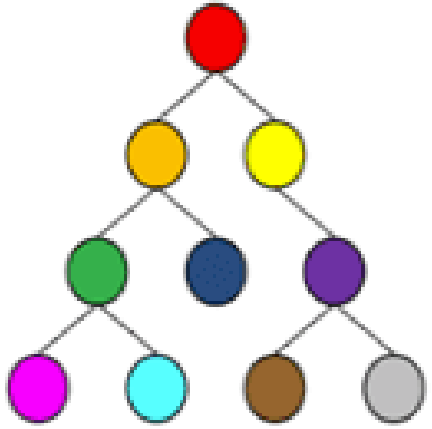
NoSQL Database

Column-Family

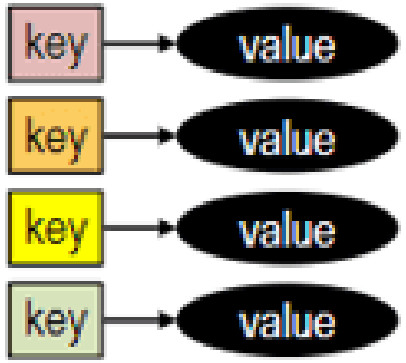
Graph



Document



Key-Value



NoSQL

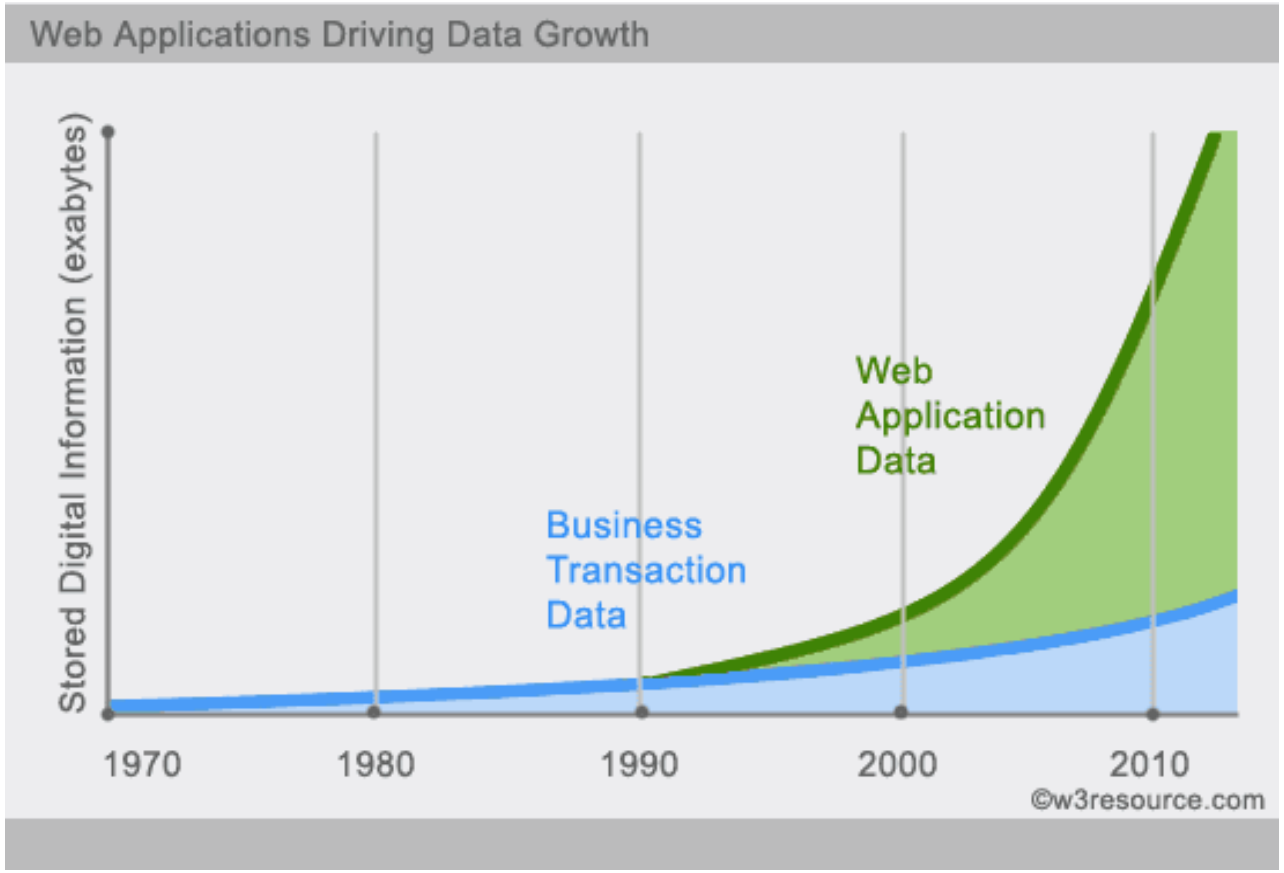
Features

- Non-relational data model
- Runs well on clusters
- Mostly open-source
- Built for the new generation Web applications
- Is schema-less

Why?

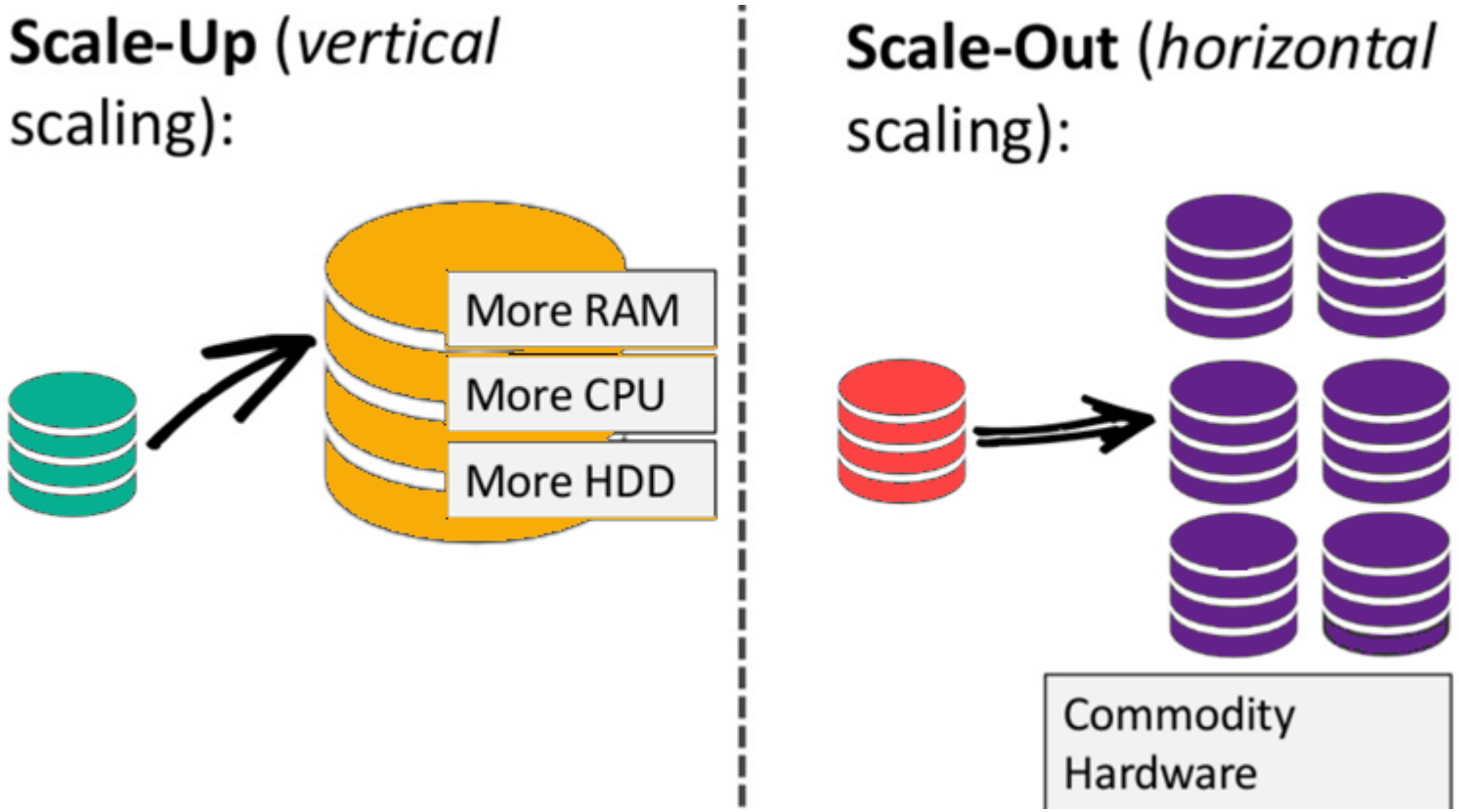
- Big data introduced new challenges and opportunities for data storage, management, analysis, and archival.

- Big data is becoming increasingly semi-structured and sparse. RDBMS databases require upfront schema definition



Why NoSQL?

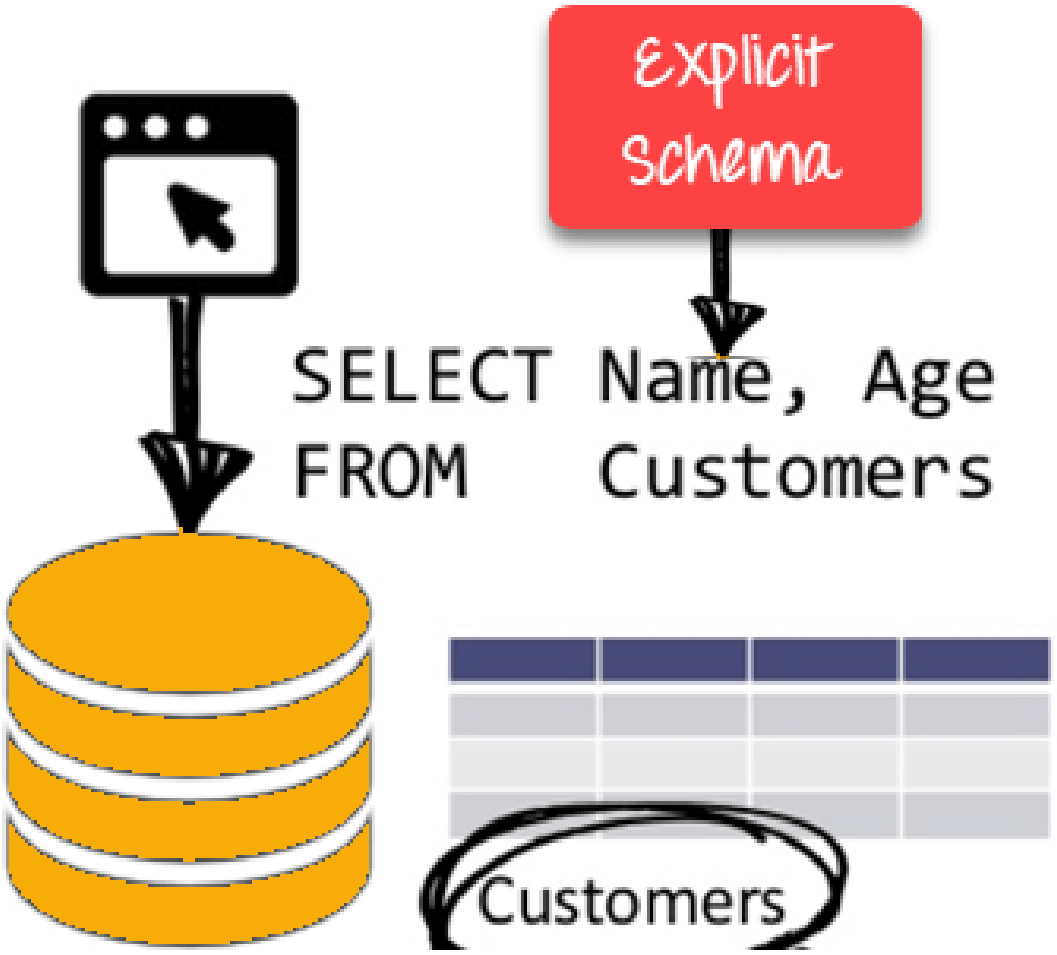
- we could "scale up" our systems by upgrading our existing hardware. This process is expensive.
- The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."



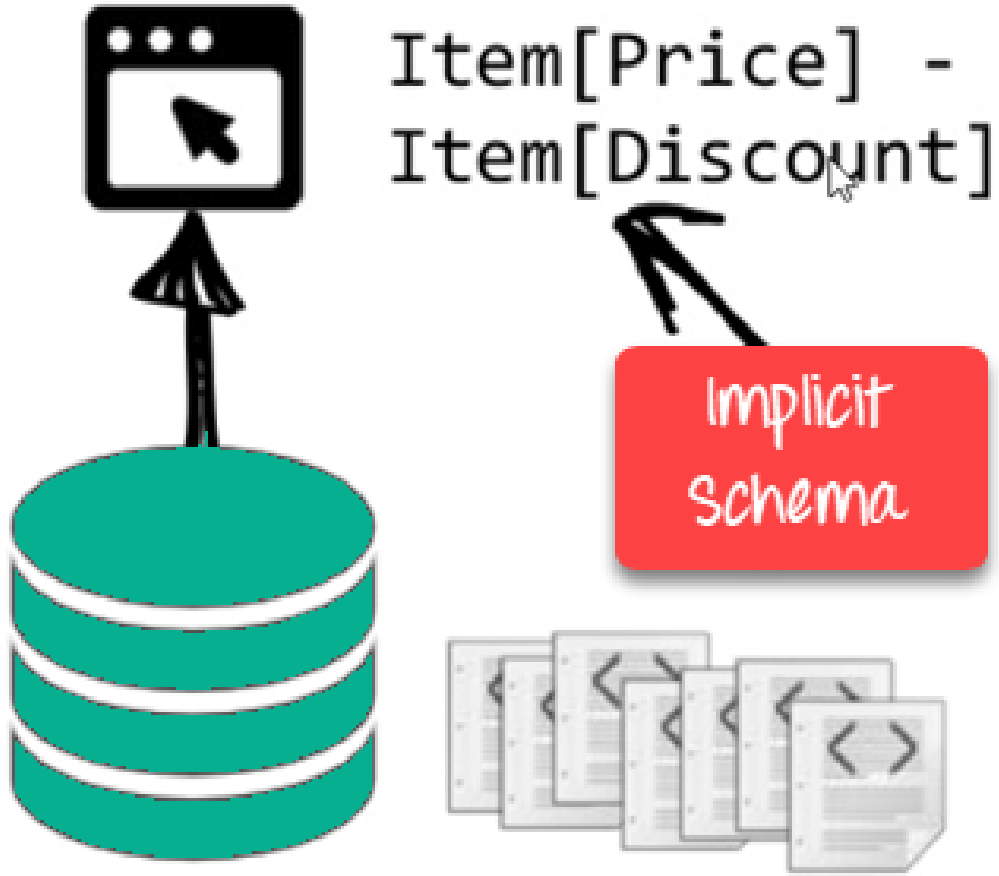
Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

RDBMS:



NoSQL DB:



RDBMS	NoSQL
• Data is stored in a relational model, with rows and columns. • A row contains information about an item while columns contain specific information, such as 'Model', 'Date of Manufacture', 'Color'. • Follows fixed schema. Meaning, the columns are defined and locked before data entry. In addition, each row contains data for each column. • Supports vertical scaling. Scaling an RDBMS across multiple servers is a challenging and time-consuming process. • Atomicity, Consistency, Isolation & Durability(ACID) Compliant	• Data is stored in a host of different databases, with different data storage models. • Follows dynamic schemas. Meaning, you can add columns anytime. • Supports horizontal scaling. You can scale across multiple servers. Multiple servers are cheap commodity hardware or cloud instances, which make scaling cost-effective compared to vertical scaling. • Not ACID Compliant.

Benefits of NoSQL data stores

- Primary and Analytic Data Source Capability
- Big Data Capability
- Continuous Availability or No Single Point of Failure
- Multi-Data Center Capability
- Easy Replication for Distributed Location-Independent Capabilities
- No Need for Separate Caching Layer
- Cloud-Ready
- High Performance with Linear Scalability
- Flexible Schema Support
- Support Key Developer Languages and Platforms
- Easy to Implement, Maintain, and Grow
- Open Source Community

Features of NoSQL

Non-relational

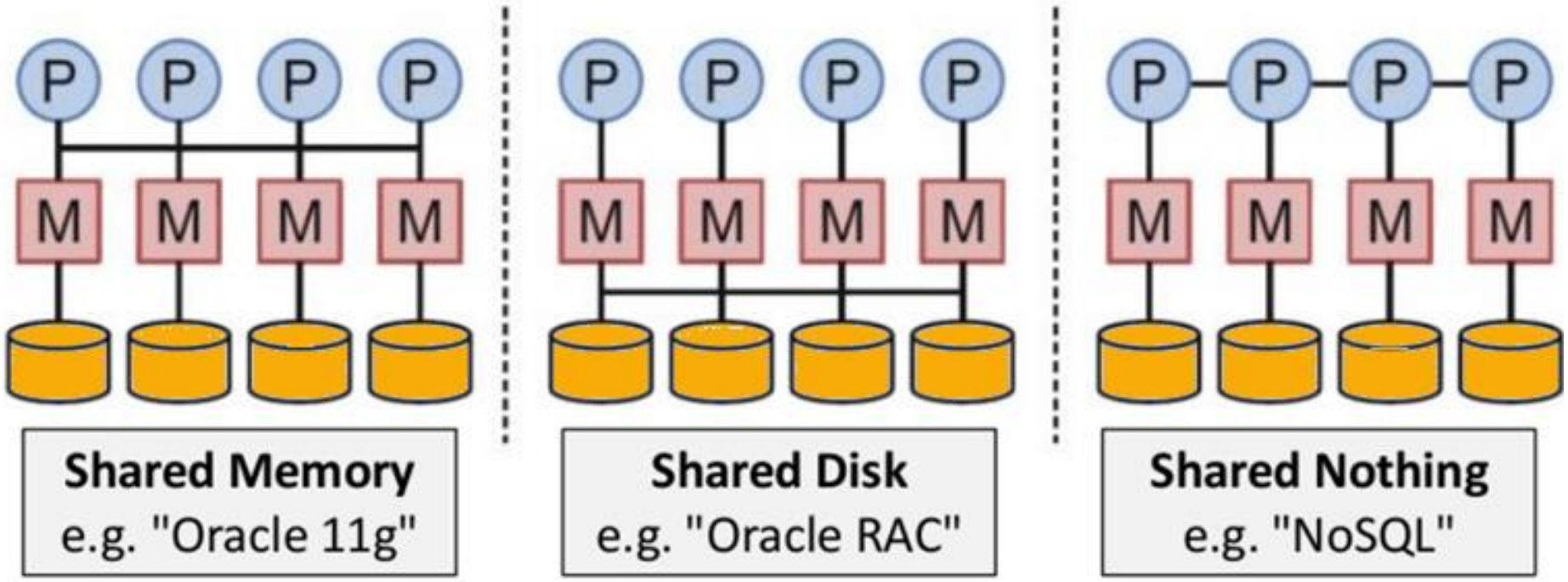
- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn't require object-relational mapping and data normalization
- No complex features like query languages, query planners,referential integrity joins, ACID

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain

Distributed

•Multiple NoSQL databases can be executed in a distributed fashion
•Offers auto-scaling and fail-over capabilities
•Often ACID concept can be sacrificed for scalability and throughput
•Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
•Only providing eventual consistency
•Shared Nothing Architecture. This enables less coordination and higher distribution.



Types of NoSQL data stores

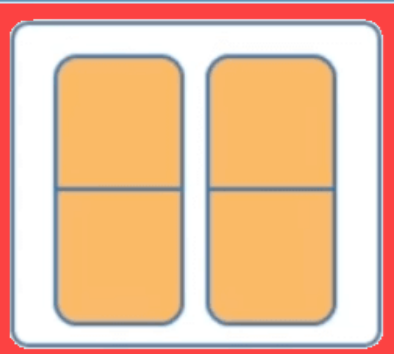
Key Value

Example:
Riak, Tokyo Cabinet, Redis server, Memcached, Scalaris

Document-Based

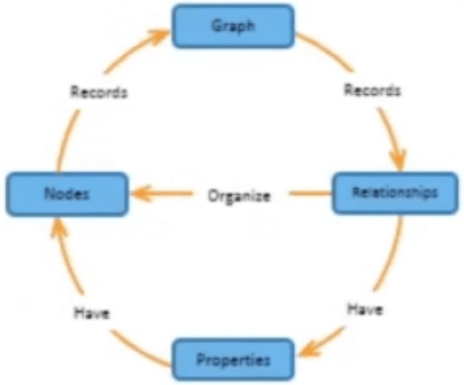
Example:
MongoDB, CouchDB, OrientDB, RavenDB

Column-Based



Example:
BigTable, Cassandra, Hbase, Hypertable

Graph-Based



Example:
Neo4J, InfoGrid, Infinite Graph, Flock DB

Key Value Pair Based

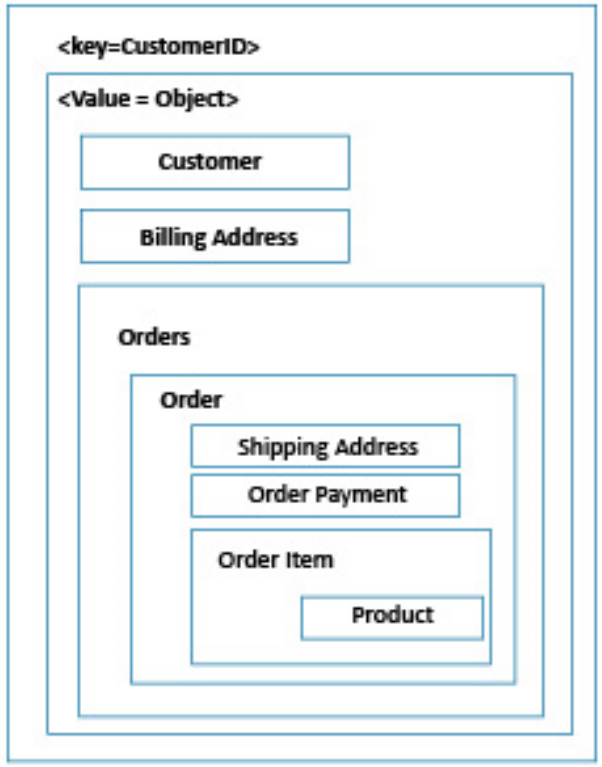
Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.

Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Redis, Dynamo, Riak are some examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.h

Key-Value



- Does not have a defined schema
- Simple to build and scale
- High performance

Advantages	Disadvantages
• Queries: You can perform a query by using the key. Even range queries on the key are usually not possible. • Schema: Key value databases have the following schema - the key is a string, the value is a blob. The client determines how to parse data. • Usages: Key value databases can access data using a key. Key-value type database suffers from major weaknesses.	• It does not provide any traditional database capabilities, such as consistency when multiple transactions are executed simultaneously. • As the volume of data increases, maintaining unique values as keys become difficult.

Column-based

Column-oriented databases work on columns and are based on BigTable paper by Google.

Every column is treated separately. Values of single column databases are stored contiguously.

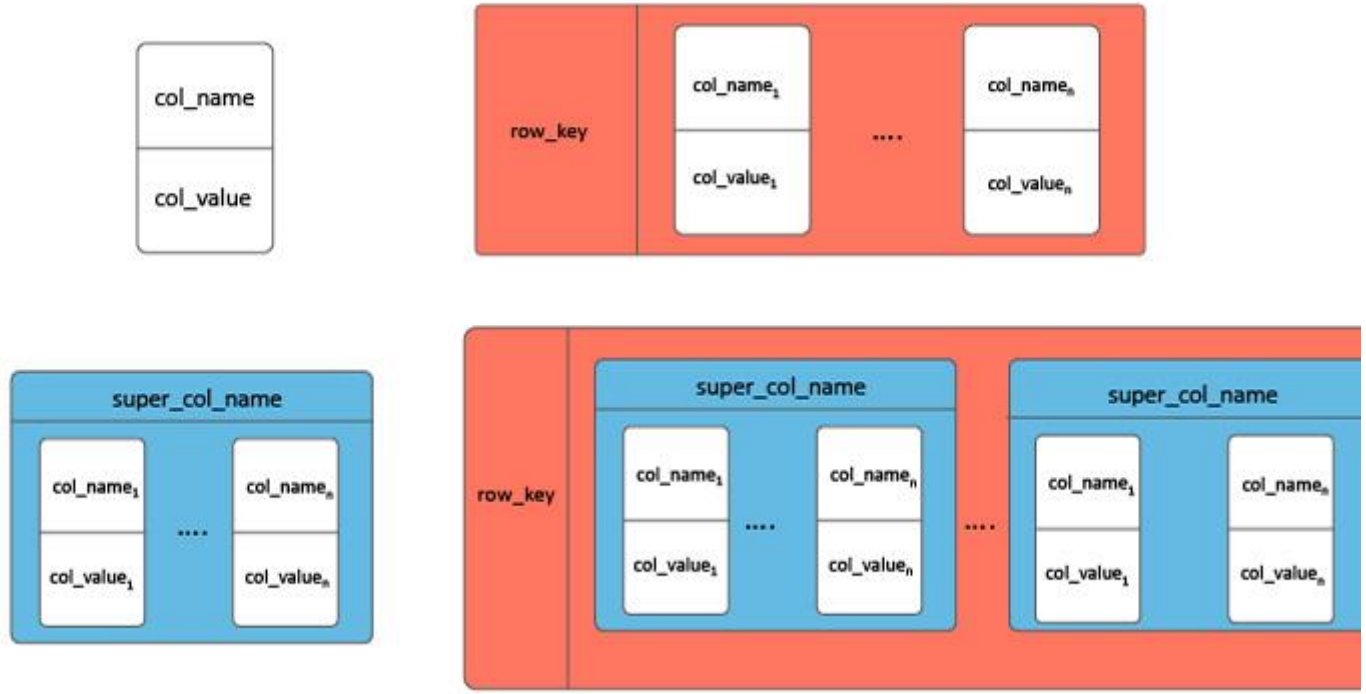
ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

High performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

HBase, Cassandra, HBase, Hypertable are examples of column based database.

Column Database

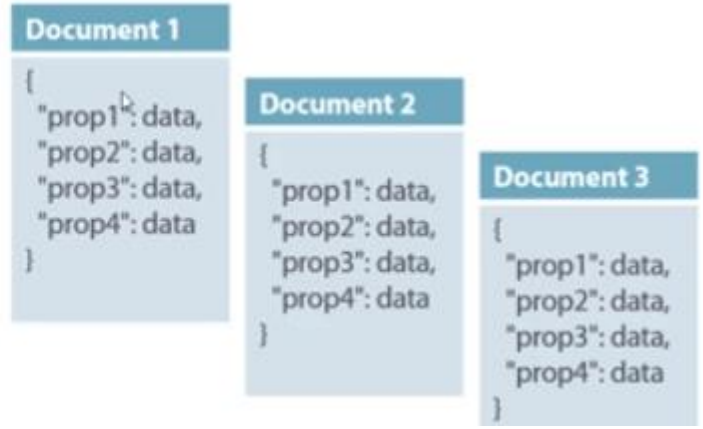
- Store data in column families as rows
- Column families are groups of related data that is accessed together
- Design goal: query efficiency



Document-Oriented

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document.

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data



The document is stored in JSON or XML formats. The value is understood by the DB and can be queried. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

Document Database

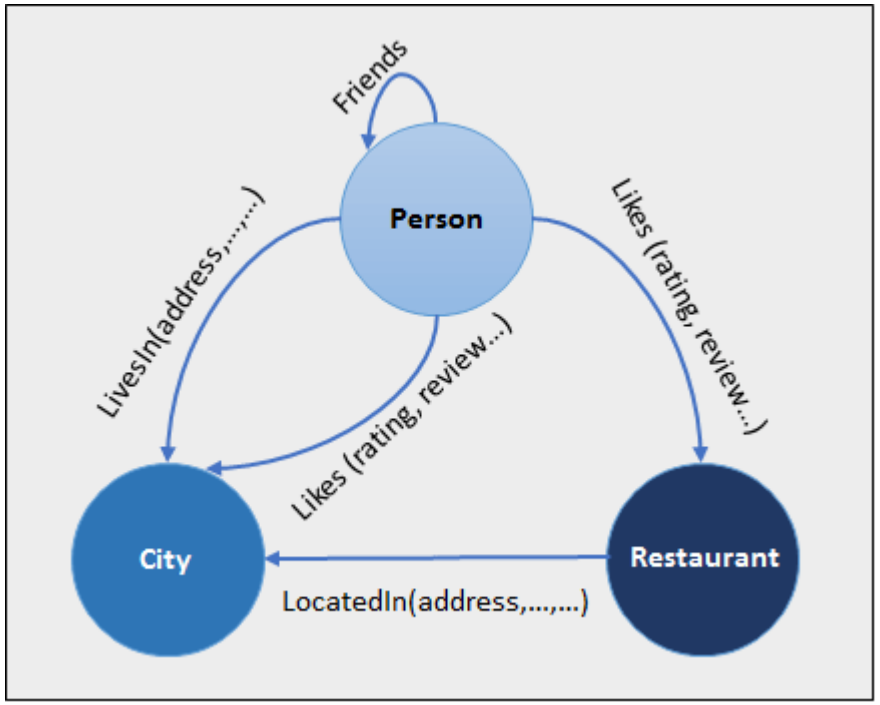
- Stores & Retrieves documents in various formats: XML, JSON, …
- Self-descriptive, hierarchical tree data
- Can be considered K-V databases, with a *document* knowledge in the value part.

e.g. MongoDB

```
Example: (MongoDB) document
{Name:“SimpliLearn",
Address:" 10685 Hazelhurst Dr, Houston, TX 77043, United States”,
Courses: [“Big Data”,”Python”,”Android”,”PMP”,”ITIL”],
Offices: [ “NYK”,”Dubai”,”BLR”],
}
```

Graph-Based

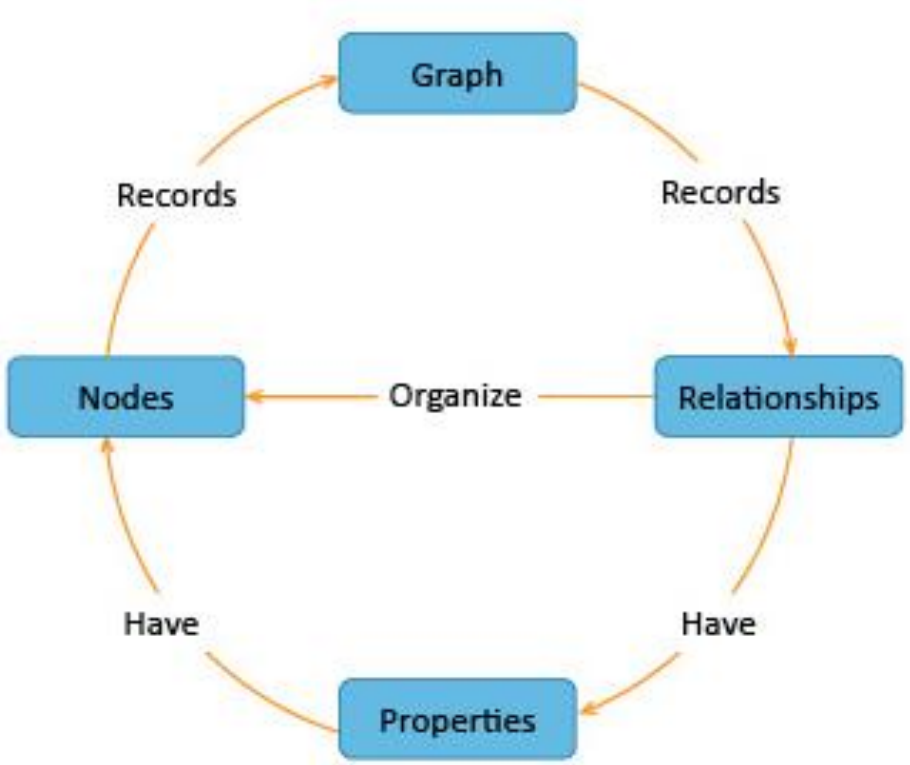
A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.

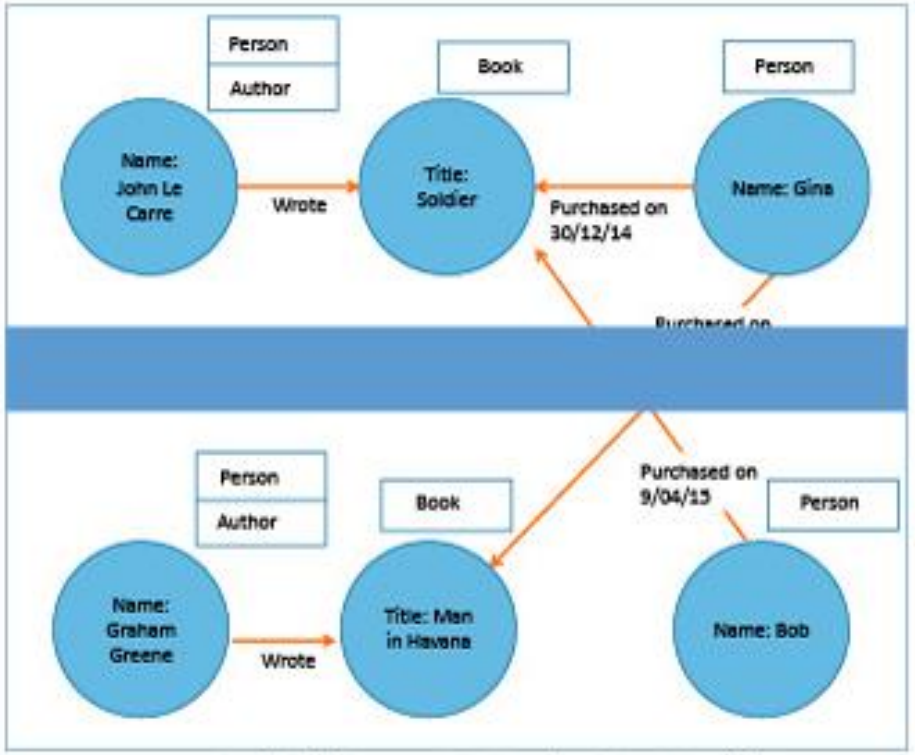


Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them. Graph base database mostly used for social networks, logistics, spatial data. Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

Graph Database

- Store data as a collection of nodes and edges
- Each *relation* can have properties
- Edges have direction





Labeled Property Graph Data Model

Query Mechanism tools for NoSQL

- The most common data retrieval mechanism is the REST-based retrieval of a value based on its key/ID with GET resource

- Document store Database offers more difficult queries as they understand the value in a key-value pair. For example, CouchDB allows defining views with MapReduce

CAP Theorem

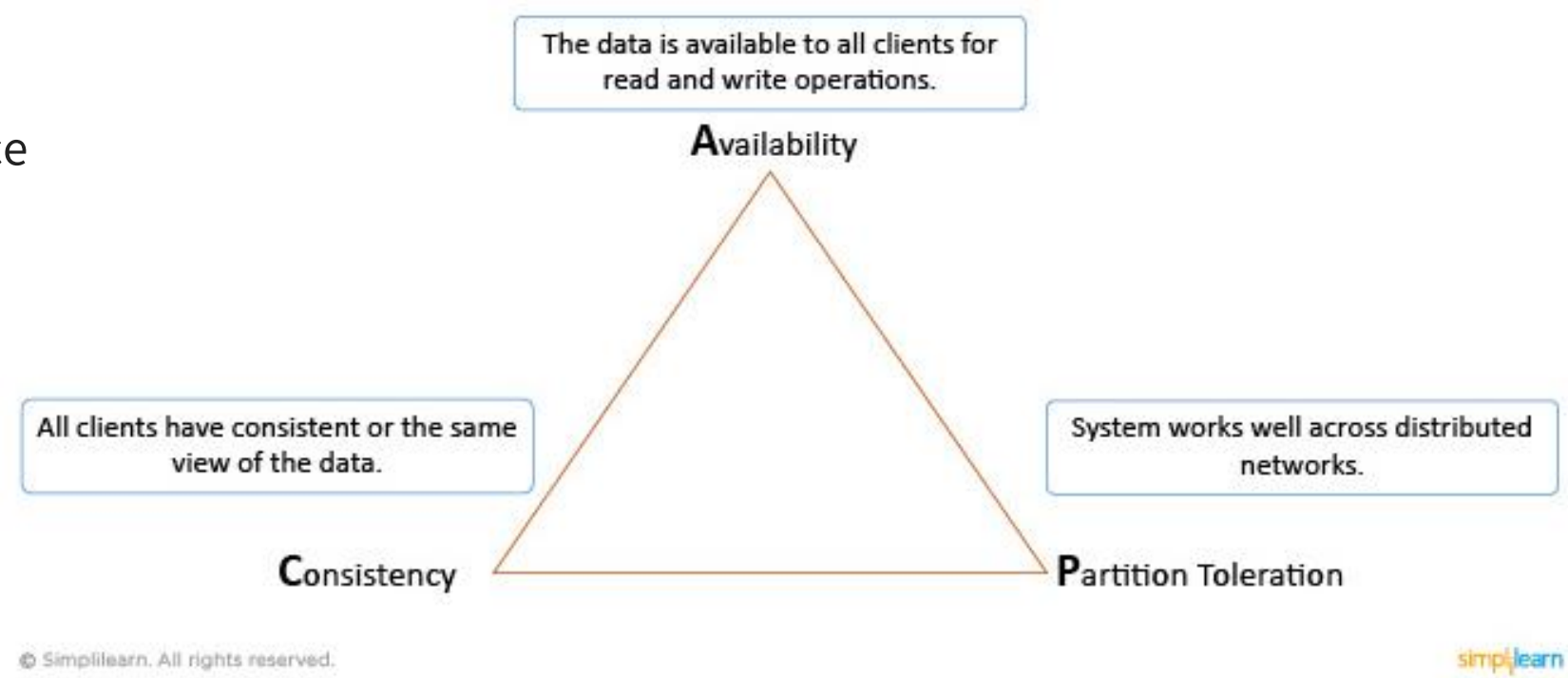
What is the CAP Theorem?

CAP theorem is also called brewer's theorem. It states that is impossible for a distributed data store to offer more than two out of three guarantees

1.Consistency
2.Availability
3.Partition Tolerance

Can have any two…

Tradeoffs…



Consistency:
The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.

Availability:
The database should always be available and responsive. It should not have any downtime.

Partition Tolerance:
Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

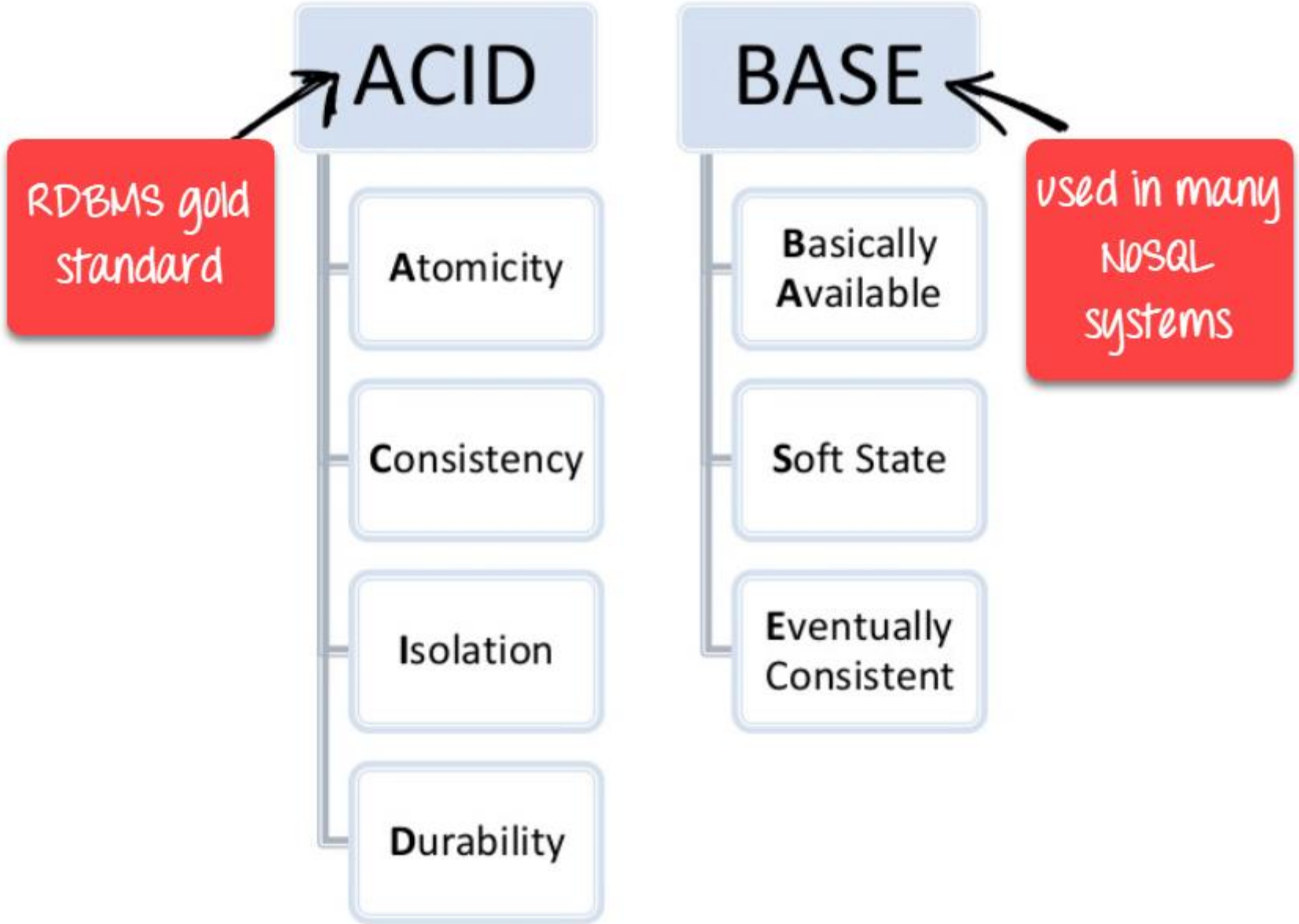
Eventual Consistency

The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability. Thus, changes made to any data item on one machine has to be propagated to other replicas.

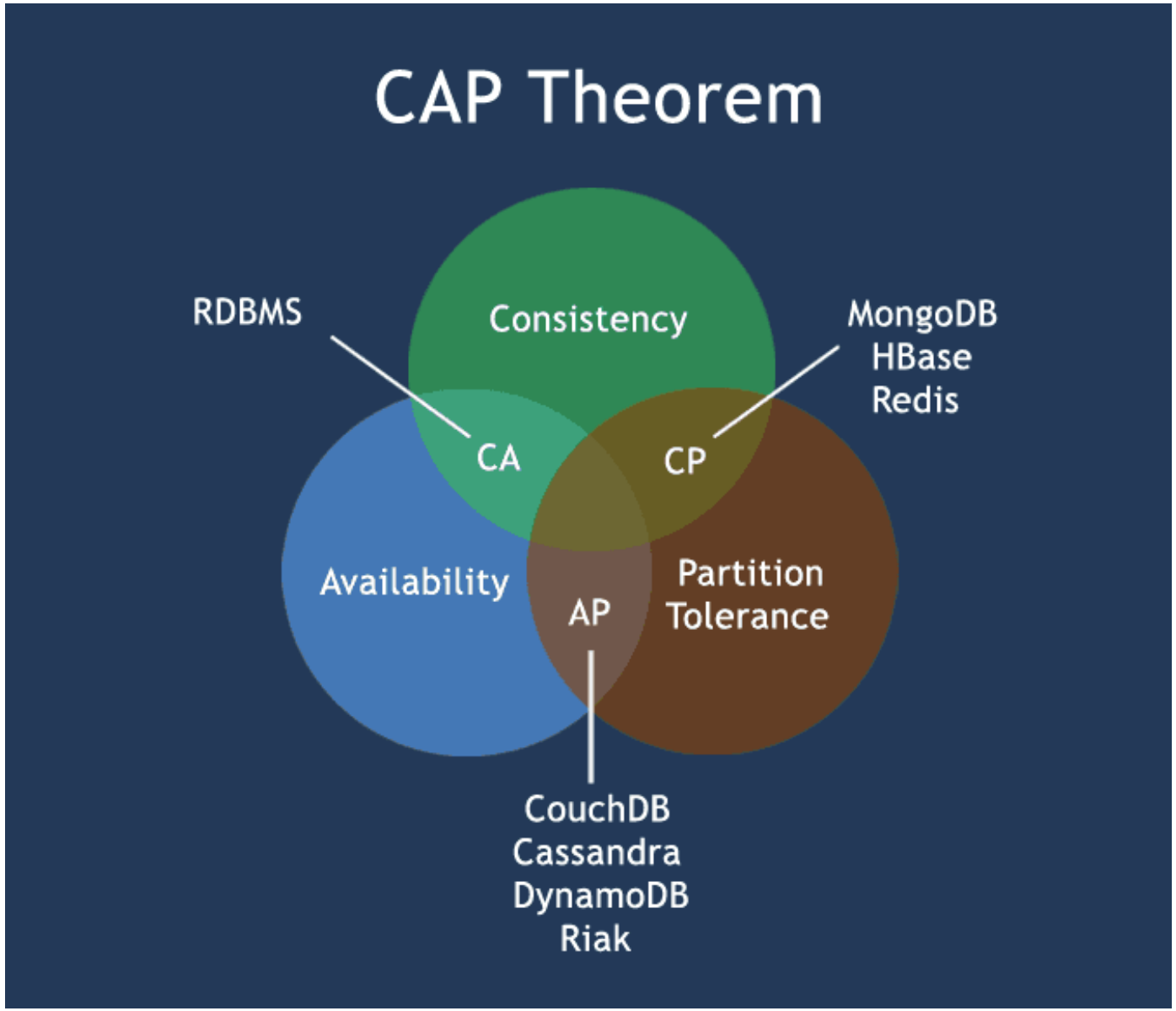
Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time. These copies may be mutually, but in due course of time, they become consistent. Hence, the name eventual consistency.

BASE: **B**asically **A**vailable, **S**oft state, **E**ventual consistency

- Basically, available means DB is available all the time as per CAP theorem
- Soft state means even without an input; the system state may change
- Eventual consistency means that the system will become consistent over time


ACID
RDBMS gold standard
Atomicity
Consistency
Isolation
Durability
BASE
used in many NOSQL systems
Basically Available
Soft State
Eventually Consistent

- Can have any two…
- Tradeoffs…



MongoDB

https://www.mongodb.com/

Document Database

Local…
https://docs.mongodb.com/manual/installation/#tutorial-installation

Cloud…
https://www.mongodb.com/cloud/atlas/lp/general?jmp=compass

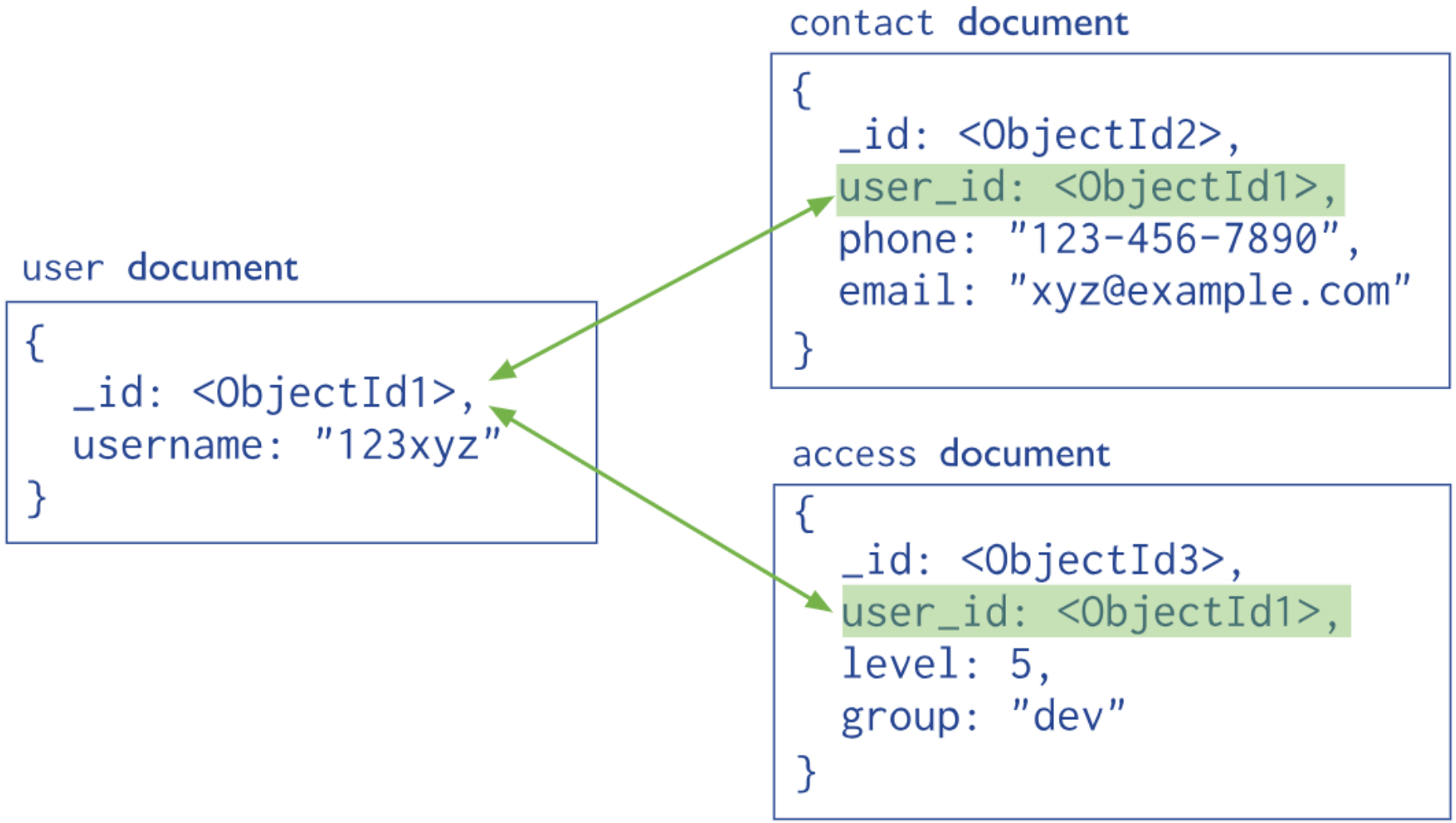
Document Structure: embedded

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
              level: 5,
              group: "dev"
            }
}
```

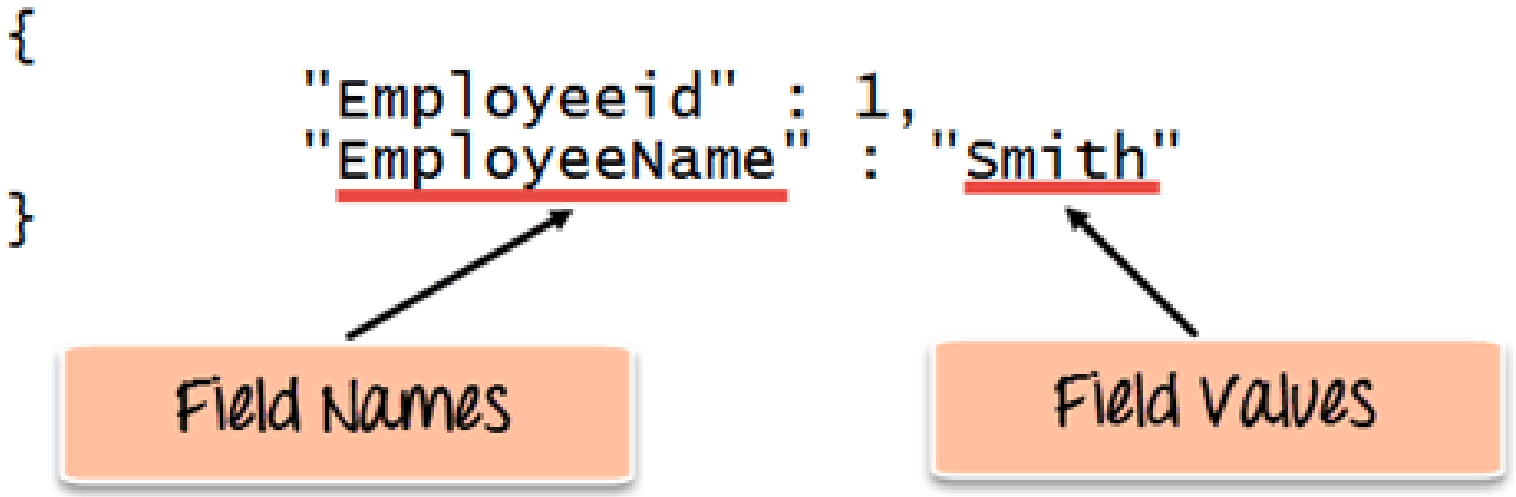
Embedded sub-document

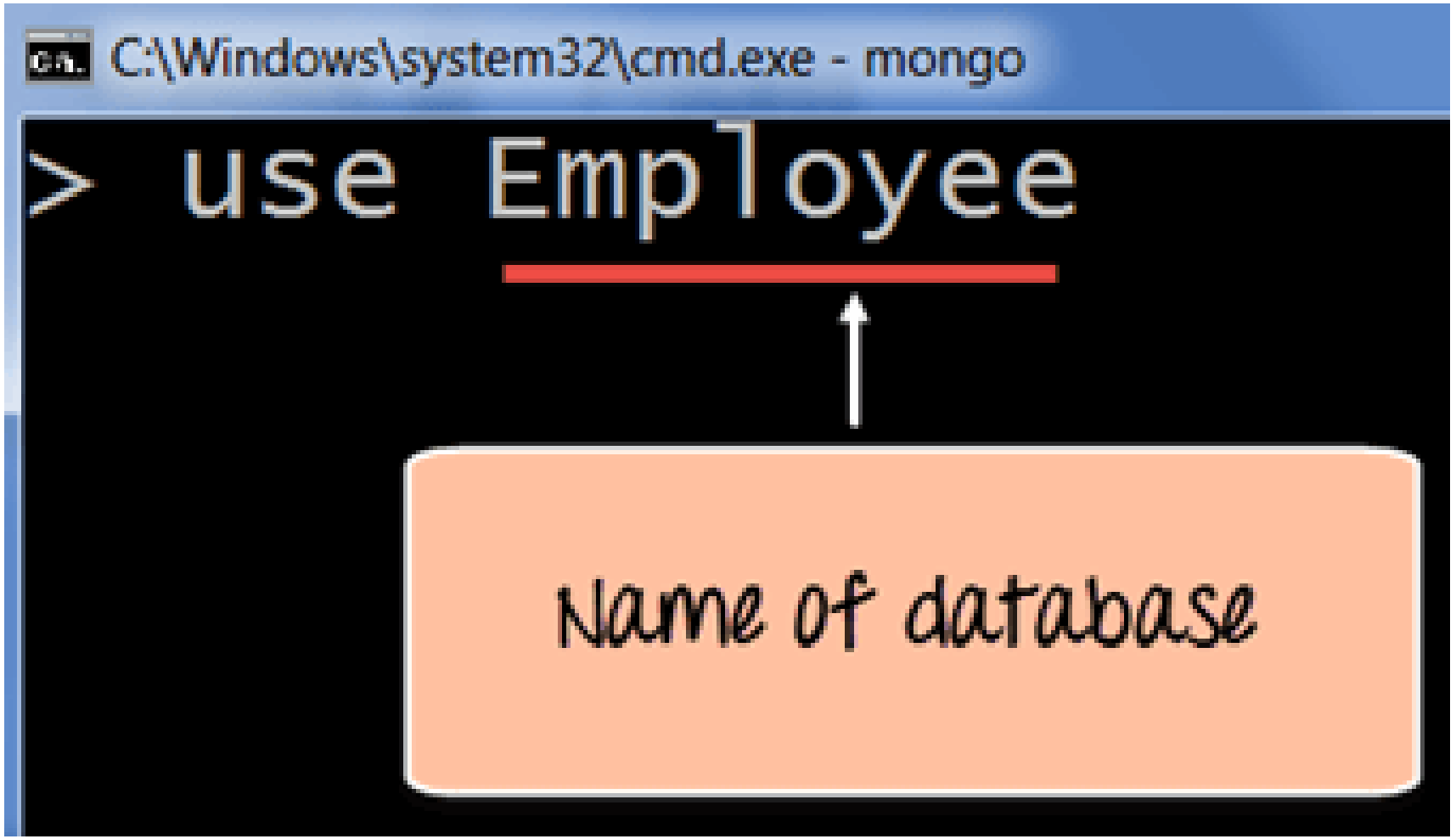
Embedded sub-document

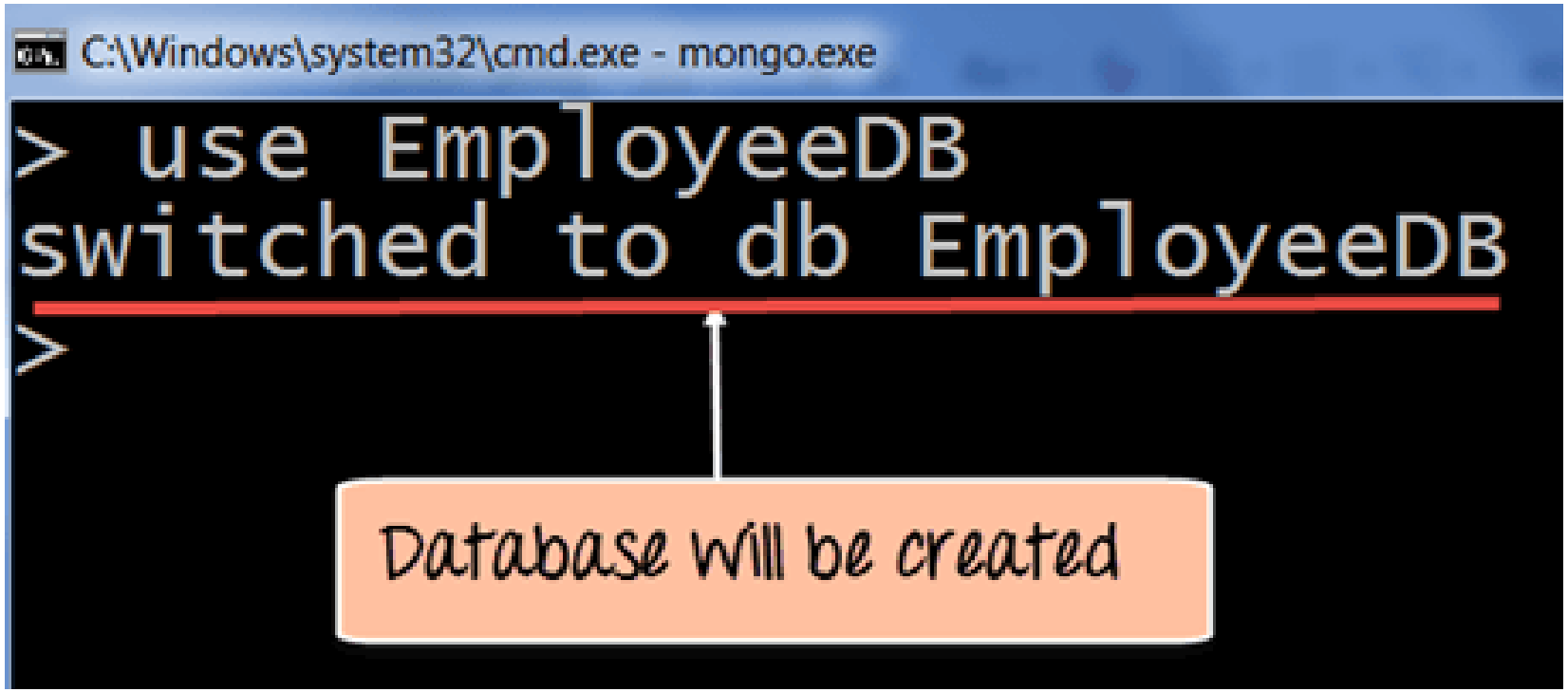
Document Structure: references

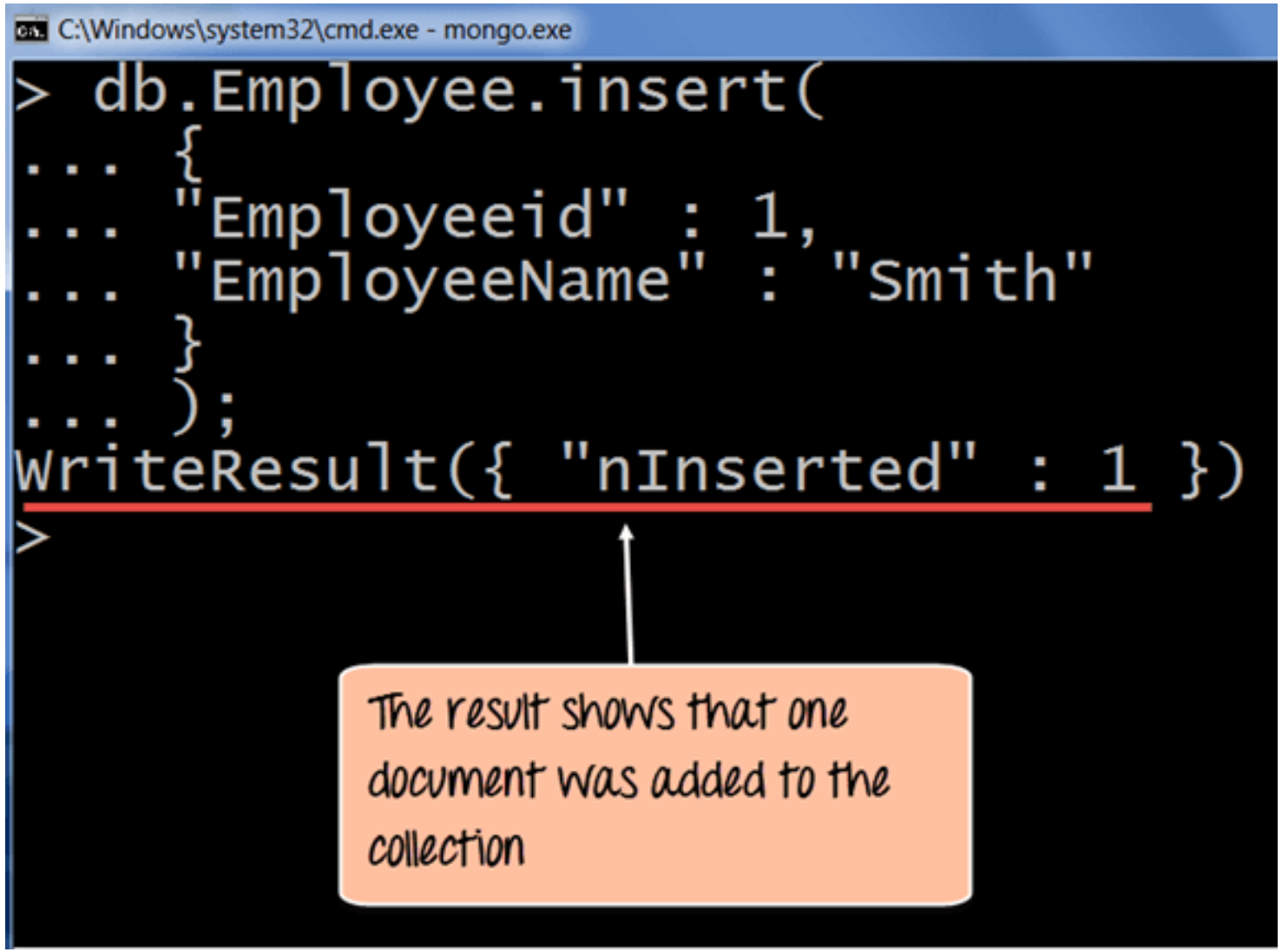


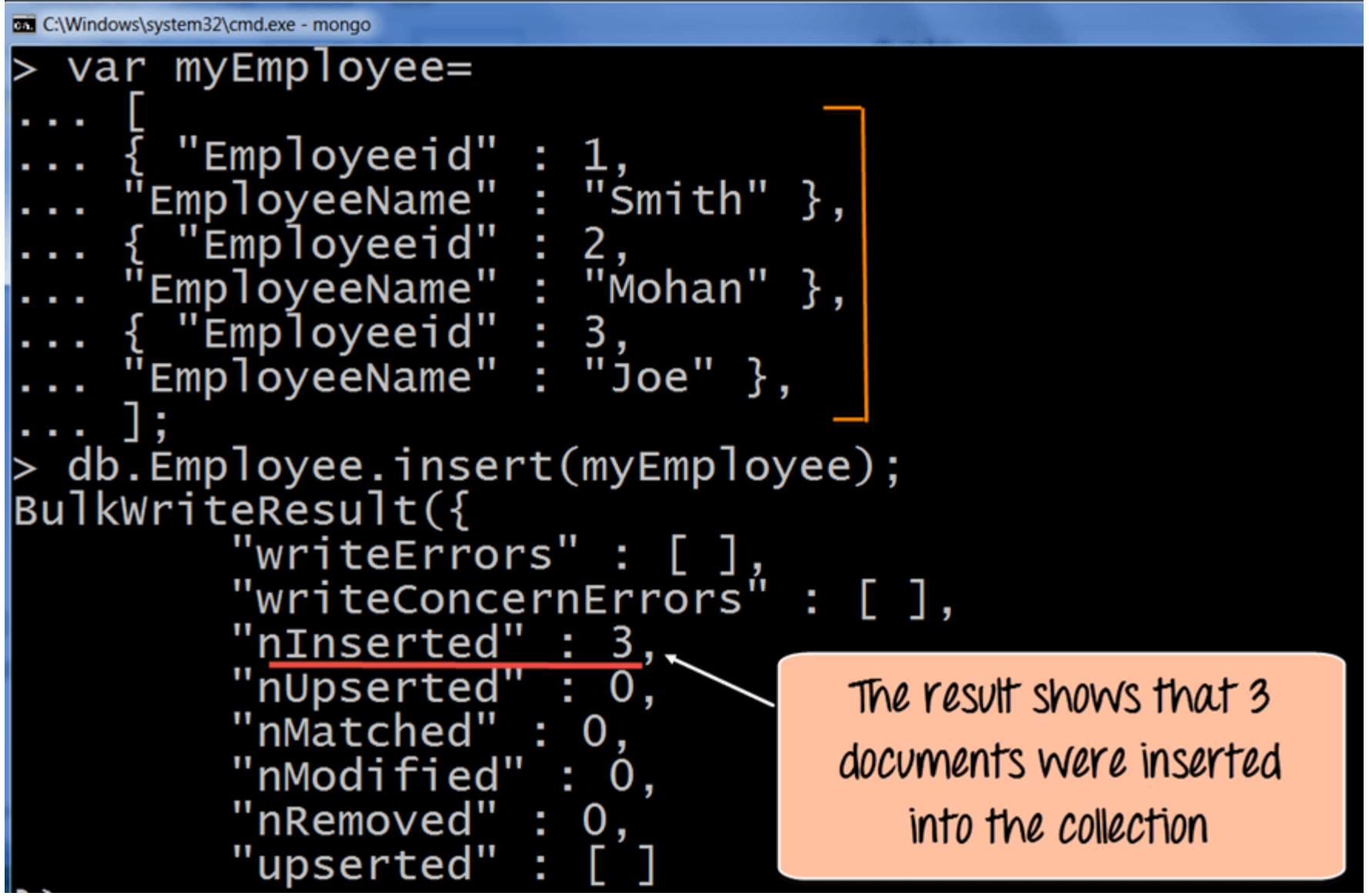
Tutorial: https://docs.mongodb.com/manual/crud/
Tutorial: https://www.guru99.com/create-read-update-operations-mongodb.html

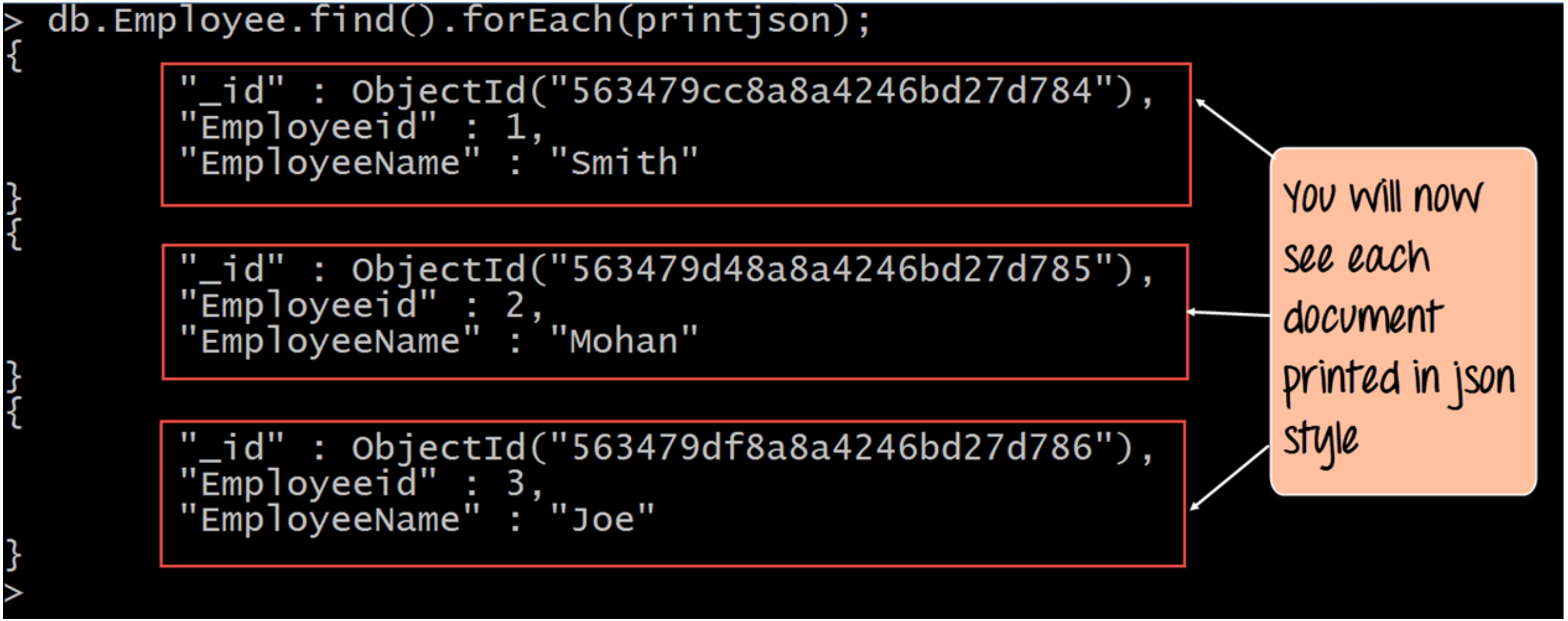


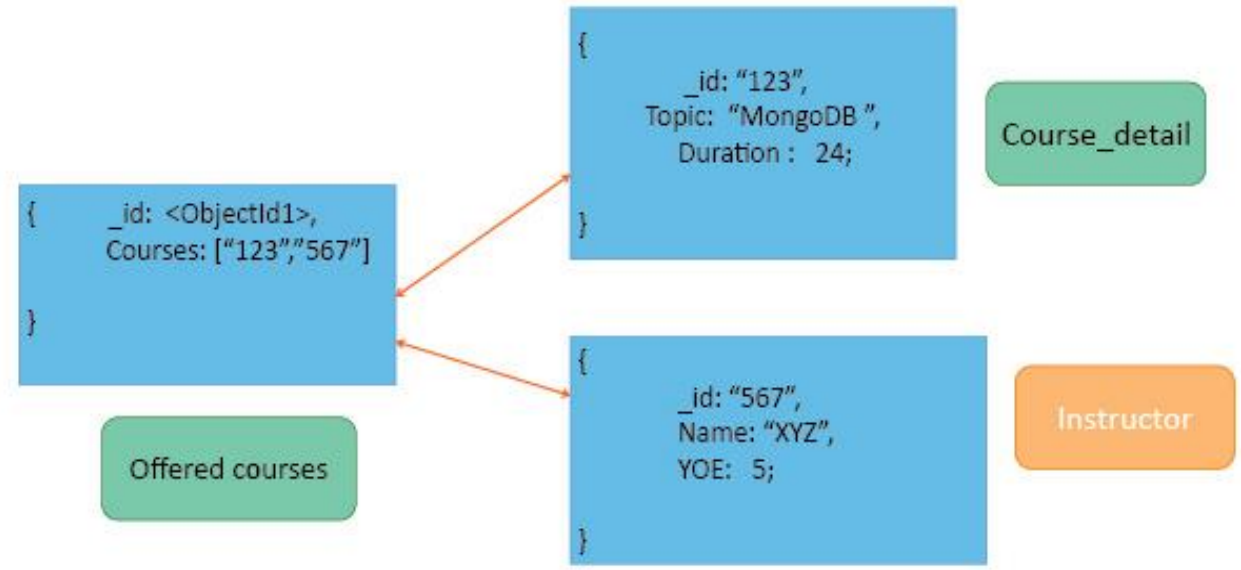

C:\Windows\system32\cmd.exe - mongo
> use EmpTOyee
Name of database


C:\Windows\system32\cmd.exe - mongo.exe
> use EmployeeDB
switched to db EmployeeDB
>
Database will be created


C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.insert(
... {
... "Employeeid" : 1,
... "EmployeeName" : "Smith"
... }
...);
WriteResult({ "nInserted" : 1 })
>
The result shows that one document was added to the collection


C:\Windows\system32\cmd.exe - mongo
> var myEmployee=
... [
... { "Employeeid" : 1,
... "EmployeeName" : "Smith" },
... { "Employeeid" : 2,
... "EmployeeName" : "Mohan" },
... { "Employeeid" : 3,
... "EmployeeName" : "Joe" },
...];
> db.Employee.insert(myEmployee);
BulkWriteResult({
"writeErrors" : [ ],
"writeConcernErrors" : [ ],
"nInserted" : 3,
"nUpserted" : 0,
"nMatched" : 0,
"nModified" : 0,
"nRemoved" : 0,
"upserted" : [ ]
The result shows that 3 documents were inserted into the collection


db.Employee.find().forEach(printjson);
{
"_id" : ObjectId("563479cc8a8a4246bd27d784"),
"Employeeid" : 1,
"EmployeeName" : "Smith"
"_id" : ObjectId("563479d48a8a4246bd27d785"),
"Employeeid" : 2,
"EmployeeName" : "Mohan"
"_id" : ObjectId("563479df8a8a4246bd27d786"),
"Employeeid" : 3,
"EmployeeName" : "Joe"
You will now see each document printed in json style


{
_id: <ObjectId1>,
Courses: ["123","567"]
}
Offered courses
{
_id: "123",
Topic: "MongoDB ",
Duration : 24;
}
Course_detail
{
_id: "567",
Name: "XYZ",
YOE: 5;
}
Instructor

MongoDB - Common Database Commands

List all databases: `show dbs`

Create a database:

Show current database: `db`

Select a database: `use dbname`

CRUD Commands

Create

```
// save one user
$ db.users.save({ name: ‘Juan' });

// save multiple users
$ db.users.save([{ name: ‘Juan'}, { name:
‘Alex' }]);
```

created both the database and collection if they did not already exist.

Read

```
// show all users
$ db.users.find();

// find a specific user
$ db.users.find({ name: ‘Juan' });
```

Update

```
db.users.update({ name: ‘Juan' }, {
name: ‘Juan Rodriguez' });
```

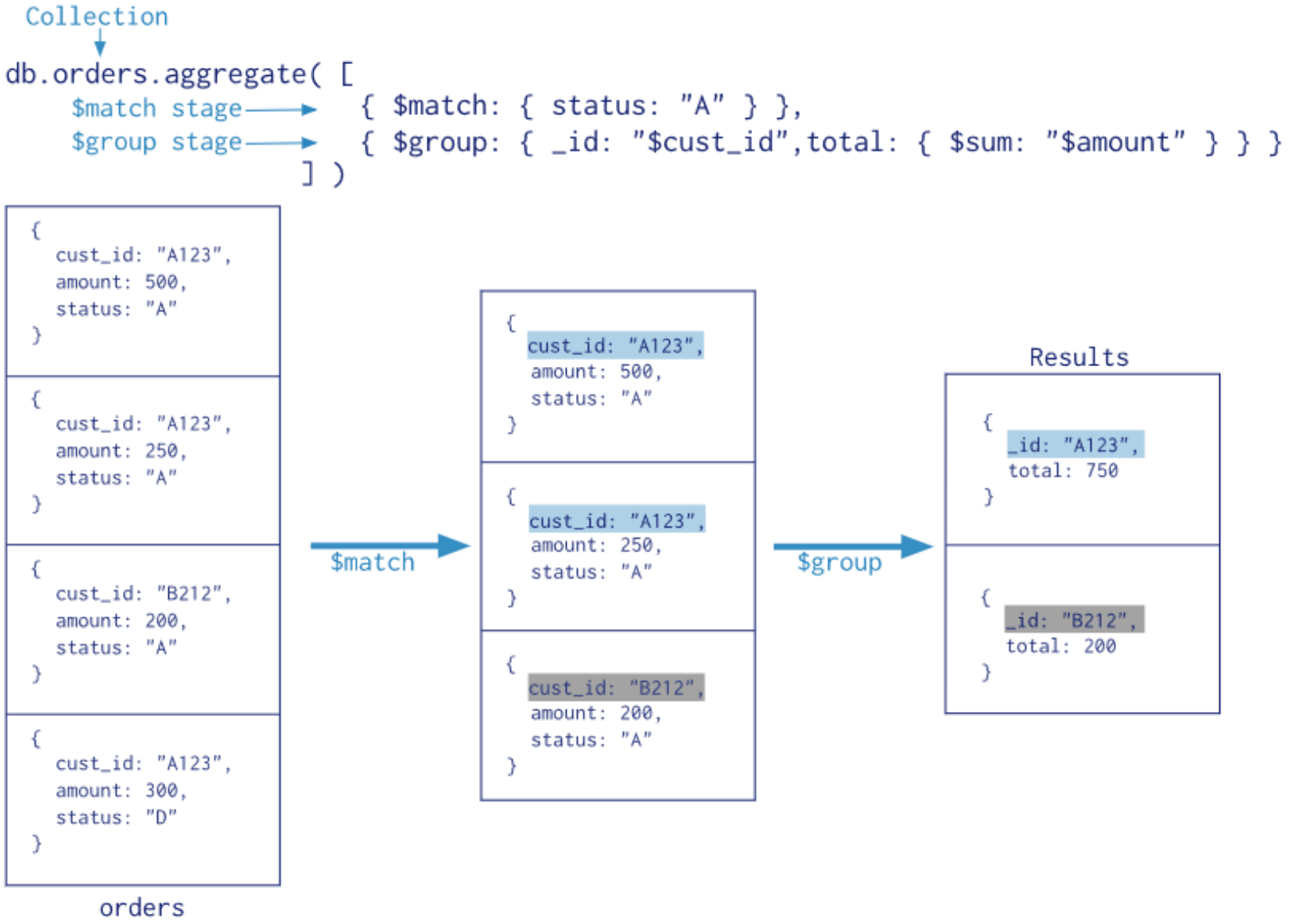
Delete

```
// remove all
db.users.remove({});

// remove one db.users.remove({ name:
‘Alex' });
```

https://docs.mongodb.com/manual/crud/

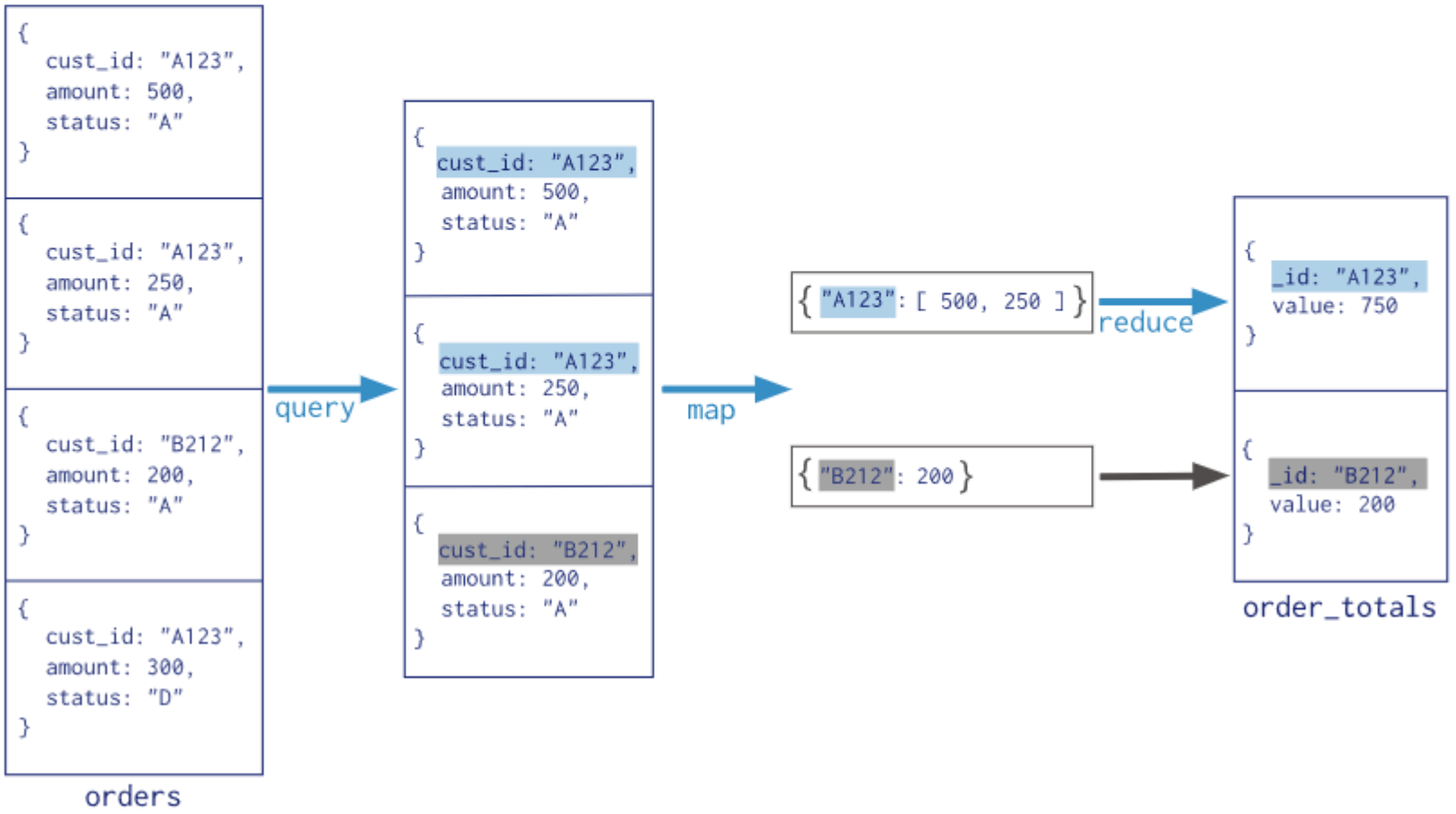
Aggregation



Map-Reduce

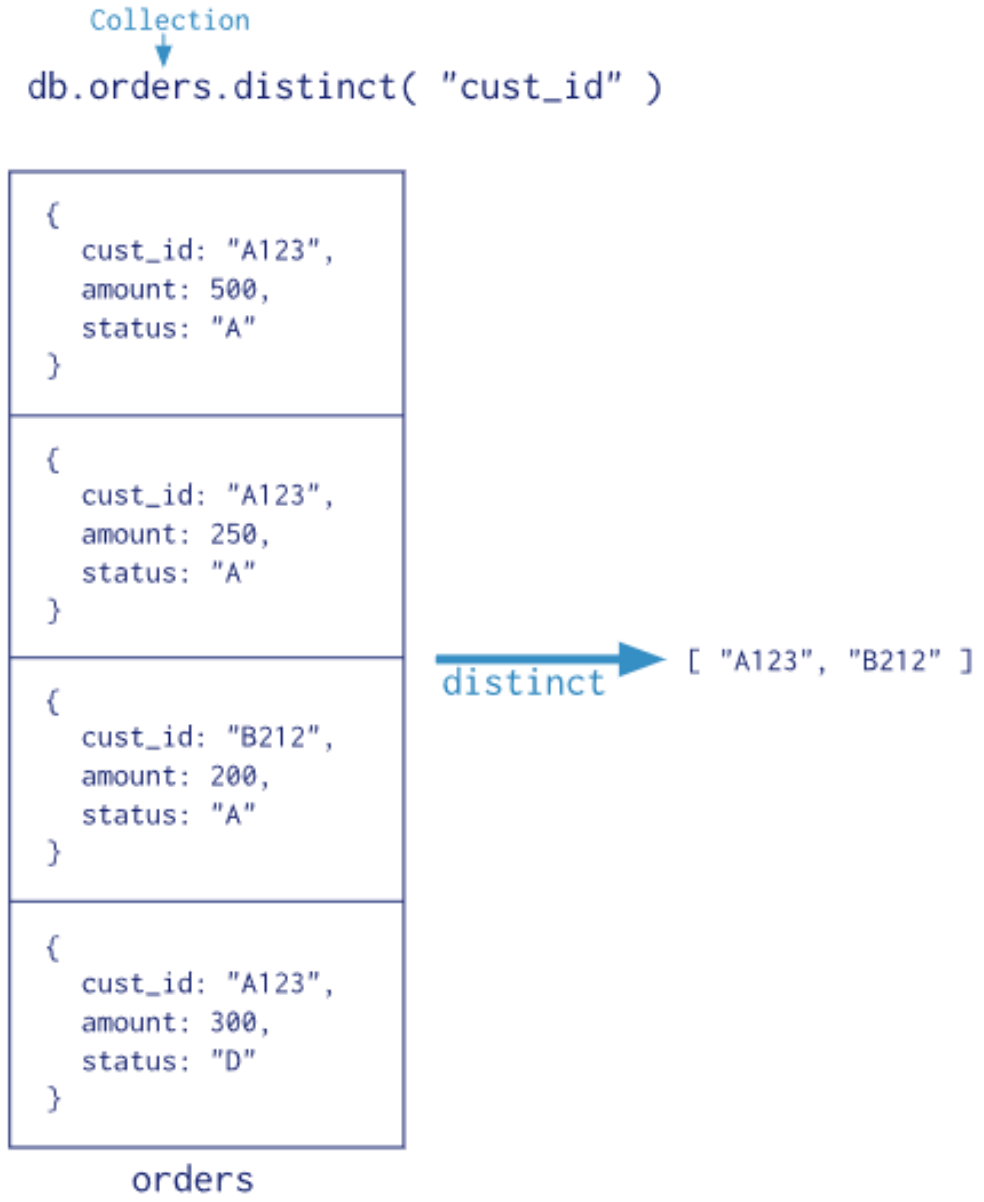
Collection

```
db.orders.mapReduce(
        map      ──►  function() { emit( this.cust_id, this.amount ); },
        reduce   ──►  function(key, values) { return Array.sum( values ) },
                      {
        query    ──►    query: { status: "A" },
        output   ──►    out: "order_totals"
                      }
                  )
```



Single Purpose Aggregation Operations

MongoDB also provides
[db.collection.estimatedDocumentCount()](#)
[db.collection.count()](#) and
[db.collection.distinct()](#).



Additional Resources

- MngoDB Analytics: Learn Aggregation by Example: Exploratory Analytics and Visualization Using Flight Data

- MongoDB for Time Series Data: Analyzing Time Series Data Using the Aggregation Framework and Hadoop

- The Aggregation Framework

- Webinar: Exploring the Aggregation Framework

- Quick Reference Cards

MongoDB & Python

https://www.anaconda.com/distribution/#download-section

Python has a native library for MongoDB. The name of the available library is “PyMongo”.

To import this, execute the following command:

```
from pymongo import MongoClient
```

```
# importing module
from pymongo import MongoClient

# creation of MongoClient
client=MongoClient()

# Connect with the portnumber and host
client = MongoClient("mongodb://localhost:27017/")

# Access database
mydatabase = client['name_of_the_database']

# Access collection of the database
mycollection=mydatabase['myTable']
```

```
# dictionary to be added in the database
rec={
title: 'MongoDB and Python',
description: 'MongoDB is no SQL database',
tags: ['mongodb', 'database', 'NoSQL'],
viewers: 104
}

# inserting the data in the database
rec = mydatabase.myTable.insert(record)
```

Querying in MongoDB :

There are certain query functions which are used to filer the data in the database. The two most commonly used functions are:

find()

```
for i in mydatabase.myTable.find({title: 'MongoDB and Python'})
        print(i)
```

count()

```
print(mydatabase.myTable.count({title: 'MongoDB and Python'}))
```

```
print(mydatabase.myTable.find({title: 'MongoDB and Python'}).count())
```

```
from pymongo import MongoClient

try:
        conn = MongoClient()
        print("Connected successfully!!!")
except:
        print("Could not connect to MongoDB")

# database name: mydatabase
db = conn.mydatabase

# Created or Switched to collection names: myTable
collection = db.myTable

# To find() all the entries inside collection name 'myTable'
cursor = collection.find()
for record in cursor:
        print(record)
```

https://www.geeksforgeeks.org/mongodb-and-python/

Next week:

- R
- Some visualizations at scale

Three weeks?

Final Project Presentations