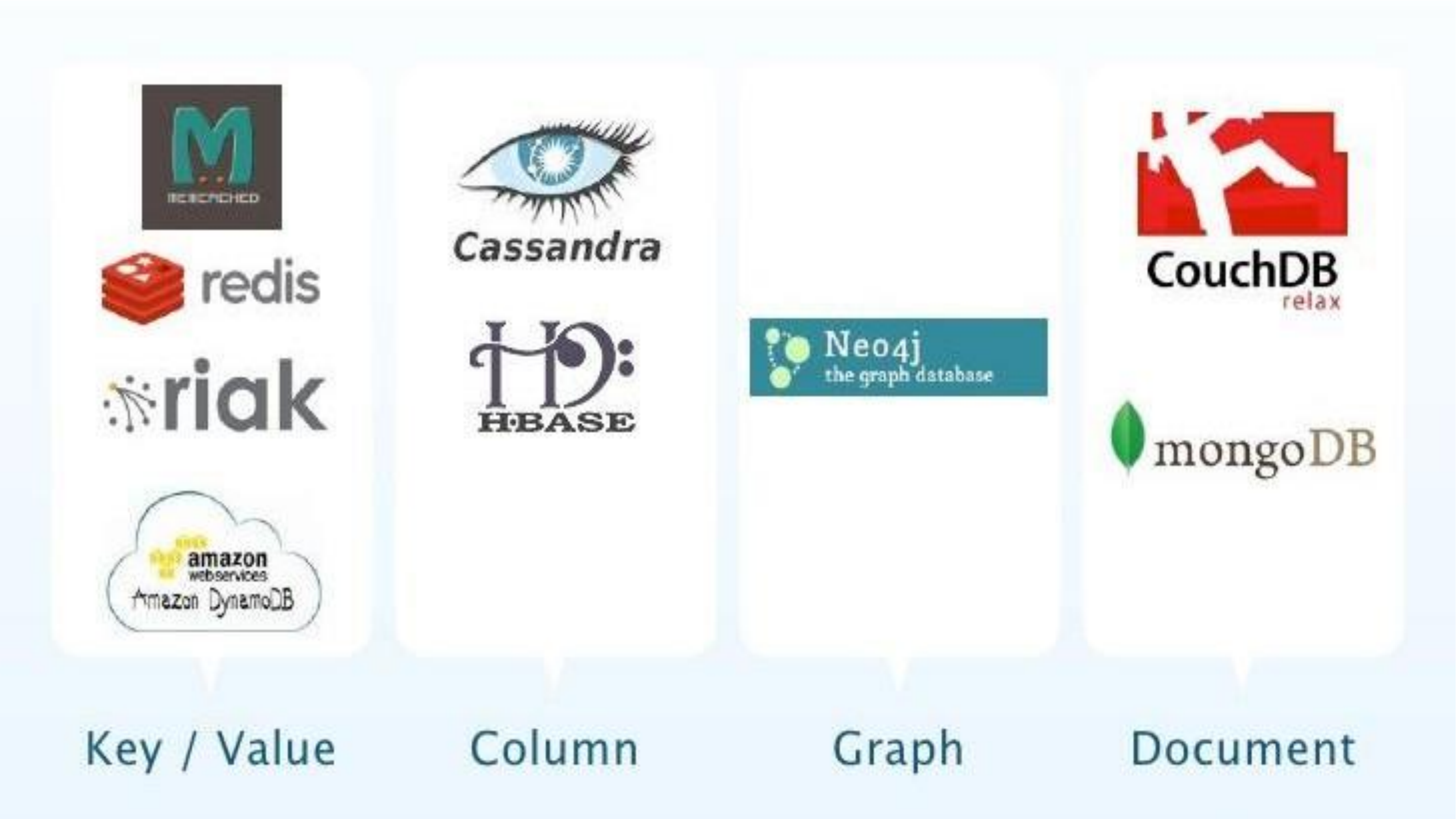
Some NoSQL databases



Some NoSQL databases

- Hundreds of thousands of **AWS** customers have chosen DynamoDB as their key-value and document database for mobile, web, gaming, ad tech, IoT, and other applications that need low-latency data access at any scale.

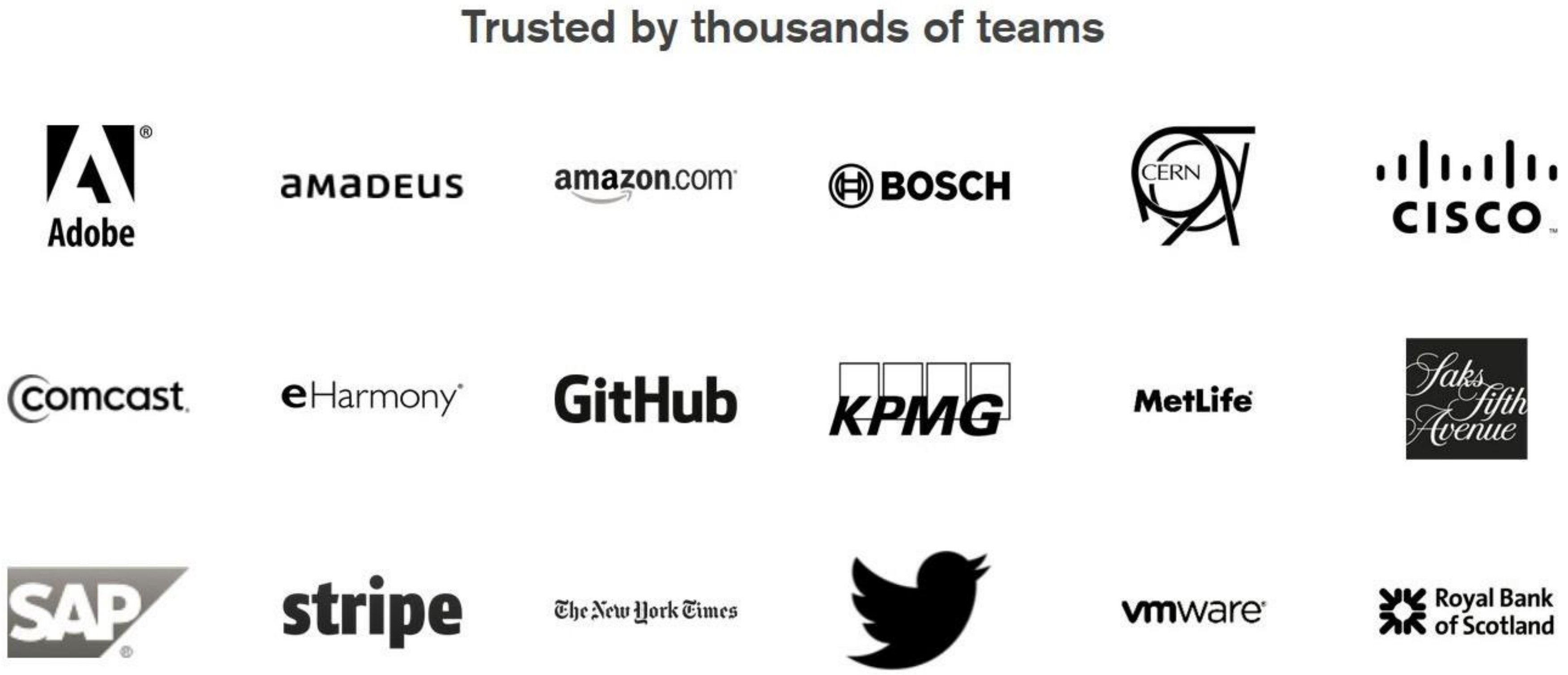
- AWS DynamoDB Fast and flexible NoSQL database service for any scale

- **Azure Cosmos DB** is a fully managed NoSQL database service for modern app development available on Microsoft Azure

- NoSQL Information on azure

https://azure.microsoft.com/en-in/overview/nosql-database/

Who's using MongoDB



Trusted by thousands of teams

NoSQL

- NoSQL, is basically a database used to manage huge sets of unstructured data, where in the data is not stored in tabular relations like relational databases. Most of the currently existing Relational Databases have failed in solving some of the complex modern problems like :
- Continuously changing nature of data - structured, semi-structured, unstructured .
- Applications now serve millions of users in different geo-locations, in different time zones and have to be up and running all the time, with data integrity.

NoSQL

- Applications are becoming more distributed with many moving towards cloud computing.
- NoSQL plays a vital role in an enterprise application which needs to access and analyze a massive set of data that is being made available on multiple virtual servers (remote based) in the cloud infrastructure and mainly when the data set is not structured.
- Hence, the NoSQL database is designed to overcome the Performance, Scalability, Data Modelling and Distribution limitations that are seen in the Relational Databases.

NoSQL Database Types

- Following are the NoSQL database types :
- **Document Databases :** In this type, key is paired with a complex data structure called as Document. Example : MongoDB
- **Graph stores :** This type of database is ususally used to store networked data. Where in we can relate data based on some existing data.
- **Key-Value stores :** These are the simplest NoSQL databases. In this each is stored with a key to identify it
- **Wide-column stores :** Used to store large data sets(store columns of data together).

  Example : Cassandra(Used in Facebook), HBase etc.

Advantages of NoSQL Databases

- **Sharding**
- In Sharding, large databases are partitioned into small, faster and easily manageable databases.
- NoSQL Databases have the Sharding feature as default. No additional efforts required. They automatically spread the data across servers, fetch the data in the fastest time from the server which is free, while maintaining the integrity of data.

Advantages of NoSQL Databases

- **Replication**
- Auto data replication is also supported in NoSQL databases by default. Hence, if one DB server goes down, data is restored using its copy created on another server in network.
-

  Integrated Caching
- Many NoSQL databases have support for Integrated Caching, where in the frequently demanded data is stored in cache to make the queries faster.

MongoDB - NoSQL Database

- MongoDB is a NoSQL database written in C++ language. Some of its drivers use the C programming language as the base.
- MongoDB is a document oriented database where it stores data in collections instead of tables. The best part of MongoDB is that the drivers are available for almost all the popular programming languages.
- NoSQL has become the first choice in database technology for developing & Deploying the applications on cloud.
-

Introduction to MongoDB

- MongoDB is a NoSQL database which stores the data in form of key-value pairs. It is an **Open Source, Document Database** which provides high performance and scalability.

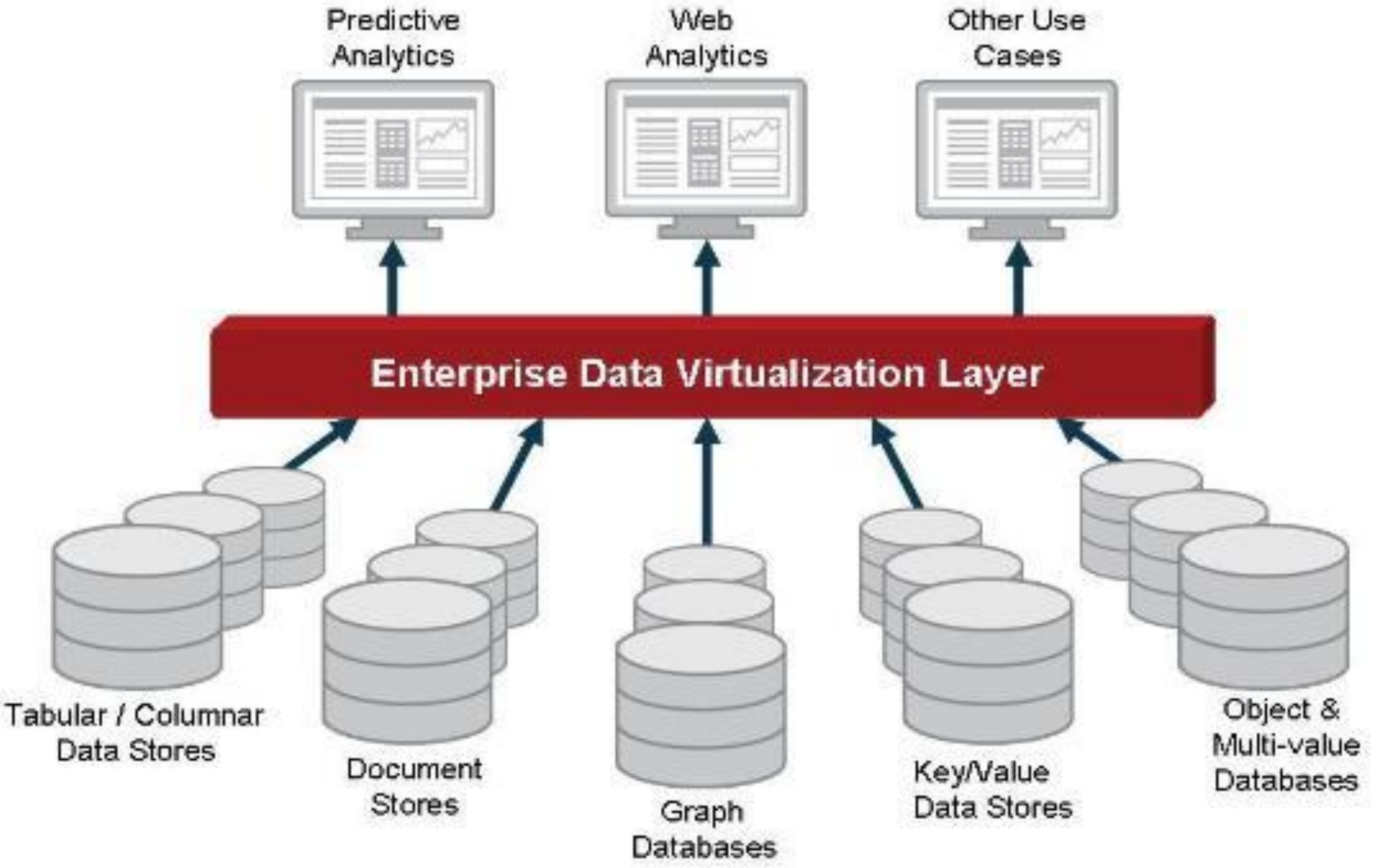
- MongoDB also provides the feature of Auto-Scaling. Since, MongoDB is a cross platform database and can be installed across different platforms like Windows, Linux etc.
- MongoDB represents JSON documents in binary-encoded format called BSON behind the scenes. BSON extends the JSON model to provide additional data types and to be efficient for encoding and decoding within different languages.

Data Stores

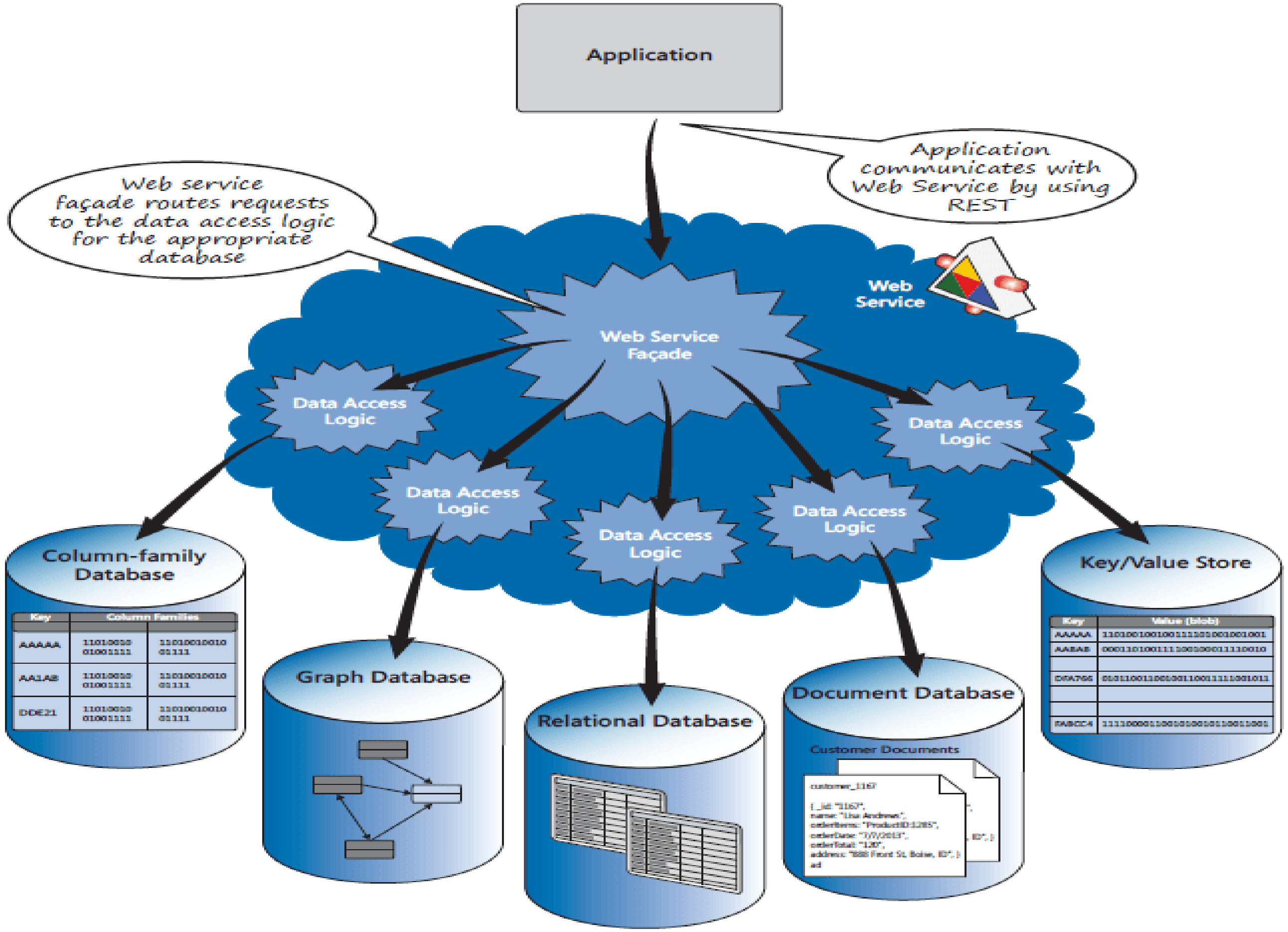
Data Stores (The Data Repository)

All base elements must be stored in the system. Derived elements, such as the employee year-to-date gross pay, may also be stored in the system. Data stores are created for each different data entity being stored. That is, when data flow base elements are grouped together to form a structural record, a data store is created for each unique structural record.

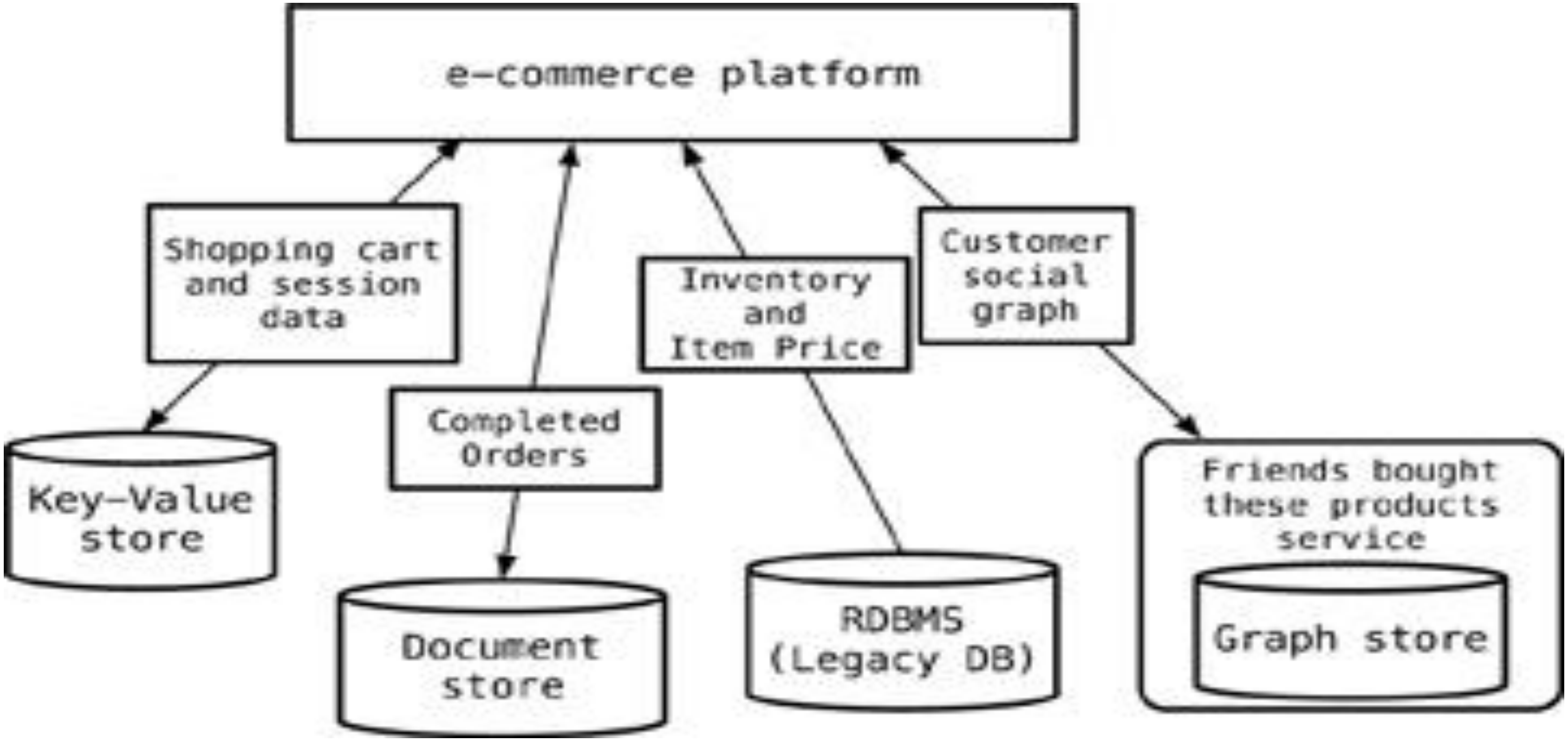
Data Stores


Predictive Analytics
Web Analytics
Other Use Cases
Enterprise Data Virtualization Layer
Tabular / Columnar Data Stores
Document Stores
Graph Databases
Key/Value Data Stores
Object & Multi-value Databases

Data Stores


Application
Application communicates with Web Service by using REST
Web service façade routes requests to the data access logic for the appropriate database
Web Service
Web Service Façade
Data Access Logic
Data Access Logic
Data Access Logic
Data Access Logic
Data Access Logic
Column-family Database
Key
Column Families
AAAAA
11010010 01001111
11010010010 01111
AA1AB
11010010 01001111
11010010010 01111
DDE21
11010010 01001111
11010010010 01111
Graph Database
Relational Database
Document Database
Customer Documents
customer_1167
{ _id: "1167",
name: "Lisa Andrews",
orderItems: "ProductID:1285",
orderDate: "7/7/2013",
orderTotal: "120",
address: "888 Front St, Boise, ID", }
ad
Key/Value Store
Key
Value (blob)
AAAAA
110100100100111101001001001
AABAB
000110100111100100001111100010
DFA766
010110011001001100111110001011
FABCC4
111100001100101001011001100

Data Stores


e-commerce platform
Shopping cart and session data
Inventory and Item Price
Customer social graph
Completed Orders
Key-Value store
Document store
RDBMS (Legacy DB)
Friends bought these products service
Graph store

What is Structured Data?

- □ Structured data is usually text files, with defined column titles and data in rows. Such data can easily be visualized in form of charts and can be processed using data mining tools.

What is Unstructured Data?

- □ Unstructured data can be anything like video file, image file, PDF, Emails etc. What does these files have in common, nothing.
- □ Structured Information can be extracted from unstructured data, but the process is time consuming. And as more and more modern data is unstructured, there was a need to have something to store such data for growing applications, hence setting path for NoSQL.

Serialization & Deserialization

- In computing, **serialization** is the process of translating a **data** structure or object state into a format that can be stored (for example, in a file or memory **data** buffer) or transmitted (for example, across a computer network) and reconstructed later.

- **Serialization** is the process of converting objects such as strings into a **JSON** format and **deserialization** is the process of converting **JSON** data into objects.

Serialization & Deserialization

- Data **serialization** is the process of converting the state of an object into a form that can be persisted or transported. In accessing **REST** service you often transfer data from client to the **REST** service or the other way around.
- **Serialization** takes an in-memory **data** structure and converts it into a series of bytes that can be stored and transferred. **Deserialization** takes a series of bytes and converts it to an in-memory **data** structure that can be consumed programmatically
- Mongodb using as a serialization format of JSON include with encoding format for storing and accessing documents. simply we can say BSON is a binary encoded format for JSON data

Relational and non-relational defined

Relational databases (RDBMS, SQL Databases)

- Example: Microsoft SQL Server, Oracle Database, IBM DB2
- Mostly used in large enterprise scenarios
- Analytical RDBMS (OLAP, MPP) solutions are Analytics Platform System, Teradata, Netezza

Non-relational databases (NoSQL databases)

- Example: Azure Cosmos DB, MongoDB, Cassandra
- Four categories: Key-value stores, Wide-column stores, Document stores and Graph stores

Origins

Using SQL Server, I need to index a few thousand documents and search them.

No problem. I can use Full-Text Search.

I'm a healthcare company and I need to store and analyze millions of medical claims per day.

Problem. Enter Hadoop.

Using SQL Server, my internal company app needs to handle a few thousand transactions per second.

No problem. I can handle that with a nice size server.

Now I have Pokémon Go where users can enter millions of transactions per second.

Problem. Enter NoSQL.

But most enterprise data just needs an RDBMS (89% market share – Gartner).

Main differences (Relational)

Pros

- Works with *structured data*
- Supports strict ACID transactional consistency
- Supports joins
- Built-in data integrity
- Large eco-system
- Relationships via constraints
- Limitless indexing
- Strong SQL
- OLTP and OLAP
- Most off-the-shelf applications run on RDBMS

Main differences (Relational)

Cons

- Does not scale out horizontally (concurrency and data size) – only vertically, unless use sharding
- Data is normalized, meaning lots of joins, affecting speed
- Difficulty in working with semi-structured data
- Schema-on-write
- Cost

Main differences (Non-relational/NoSQL)

Pros

- Works with *semi-structured data* (JSON, XML)
- Scales out (horizontal scaling – parallel query performance, replication)
- High concurrency, high volume random reads and writes
- Massive data stores
- Schema-free, schema-on-read
- Supports documents with different fields
- High availability
- Cost
- Simplicity of design: no “impedance mismatch”
- Finer control over availability
- Speed, due to not having to join tables

Main differences (Non-relational/NoSQL)

Cons

- Weaker or eventual consistency (BASE) instead of ACID
- Limited support for joins, does not support star schema
- Data is denormalized, requiring mass updates (i.e. product name change)
- Does not have built-in data integrity (must do in code)
- No relationship enforcement
- Limited indexing
- Weak SQL
- Limited transaction support
- Slow mass updates
- Uses 10-50x more space (replication, denormalized, documents)
- Difficulty tracking schema changes over time
- Most NoSQL databases are still too immature for reliable enterprise operational applications

ACID (RDBMS) vs BASE (NoSQL)

ATOMICITY: All data and commands in a transaction succeed, or all fail and roll back

CONSISTENCY: All committed data must be consistent with all data rules including constraints, triggers, cascades, atomicity, isolation, and durability

ISOLATION: Other operations cannot access data that has been modified during a transaction that has not yet completed

DURABILITY: Once a transaction is committed, data will survive system failures, and can be reliably recovered after an unwanted deletion

Needed for bank transactions

Basically Available: Guaranteed Availability

Soft-state: The state of the system may change, even without a query (because of node updates)

Eventually Consistent: The system will become consistent over time

Ok for web page visits

ACID	BASE
Strong Consistency	Weak Consistency – stale data OK
Isolation	Last Write Wins
Transaction	Programmer Managed
Available/Consistent	Available/Partition Tolerant
Robust Database/Simpler Code	Simpler Database, Harder Code

Relational stores

Data stored in tables.

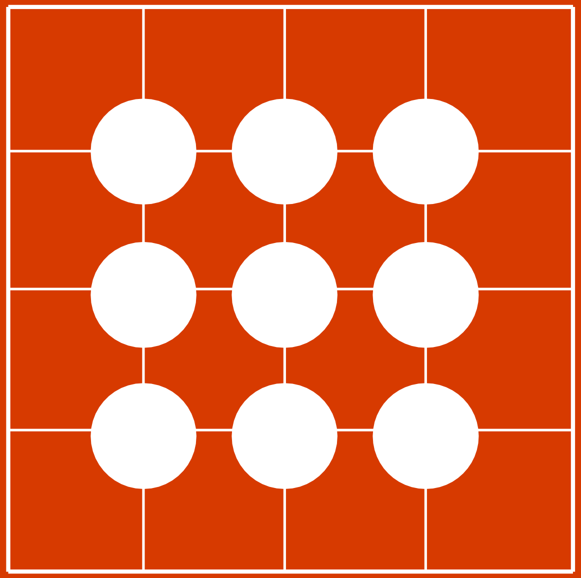
Tables contain some number of columns, each of a type.

A schema describes the columns each table can have.

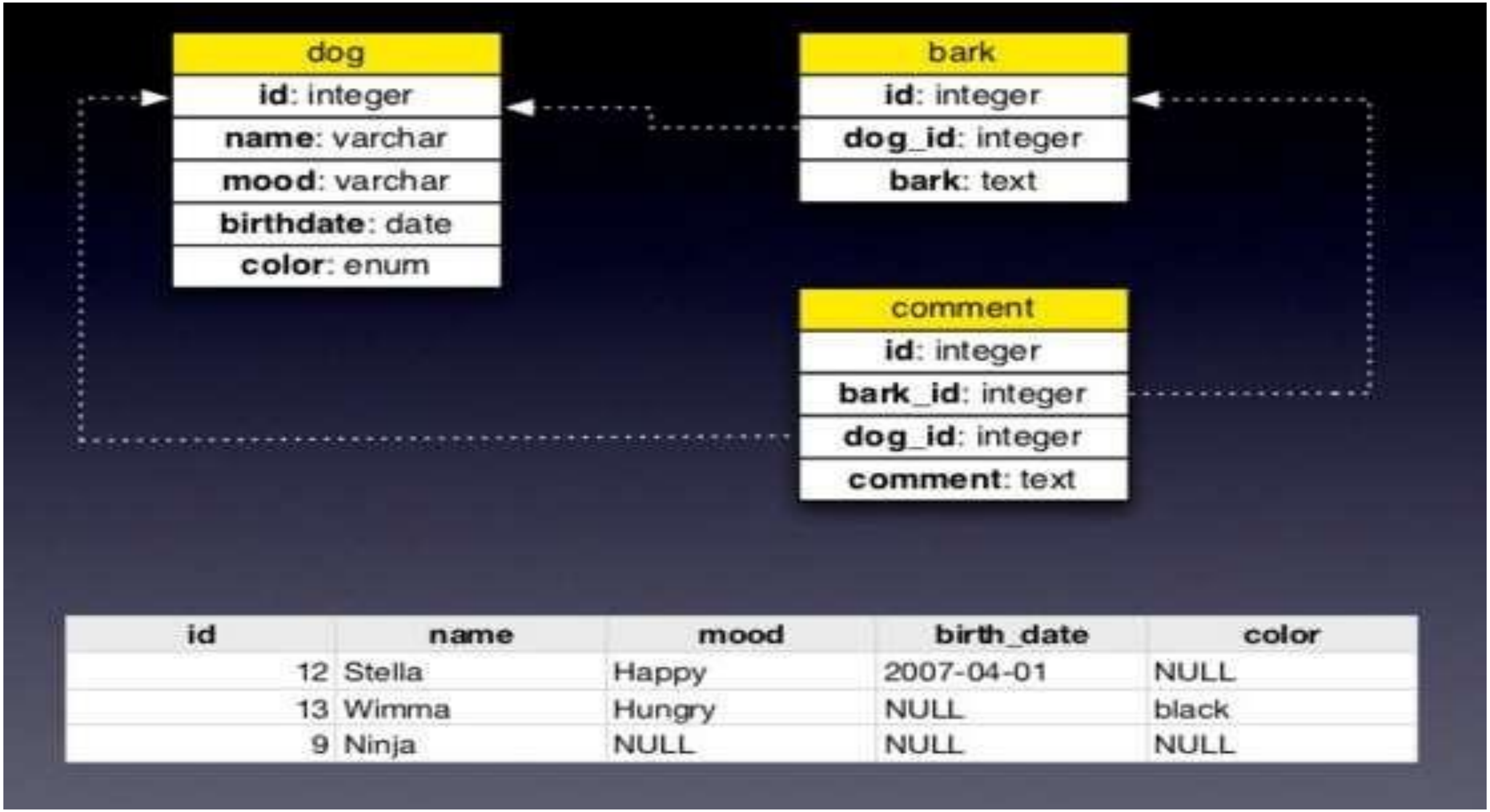
Every table’s data is stored in one or more rows.

Each row contains a value for every column in that table.

Rows aren’t kept in any particular order.



Relational stores



dog
id: integer
name: varchar
mood: varchar
birthdate: date
color: enum

bark
id: integer
dog_id: integer
bark: text

comment
id: integer
bark_id: integer
dog_id: integer
comment: text

id	name	mood	birth_date	color
12	Stella	Happy	2007-04-01	NULL
13	Wimma	Hungry	NULL	black
9	Ninja	NULL	NULL	NULL

Key-value stores

Key-value stores offer very high speed via the least complicated data model—anything can be stored as a value, as long as each value is associated with a key or name.

What is Document based storage?

- A Document is nothing but a data structure with name-value pairs like in JSON. It is very easy to map any custom Object of any programming language with a MongoDB Document.
  For example : Studentobject has
  attributes name, rollno and subjects, where subjects is a List.

Document for Student in MongoDB will be like :

```
{
        name : "Stduytonight",
        rollno : 1,
        subjects : ["C Language", "C++", "Core Java"]
}
```

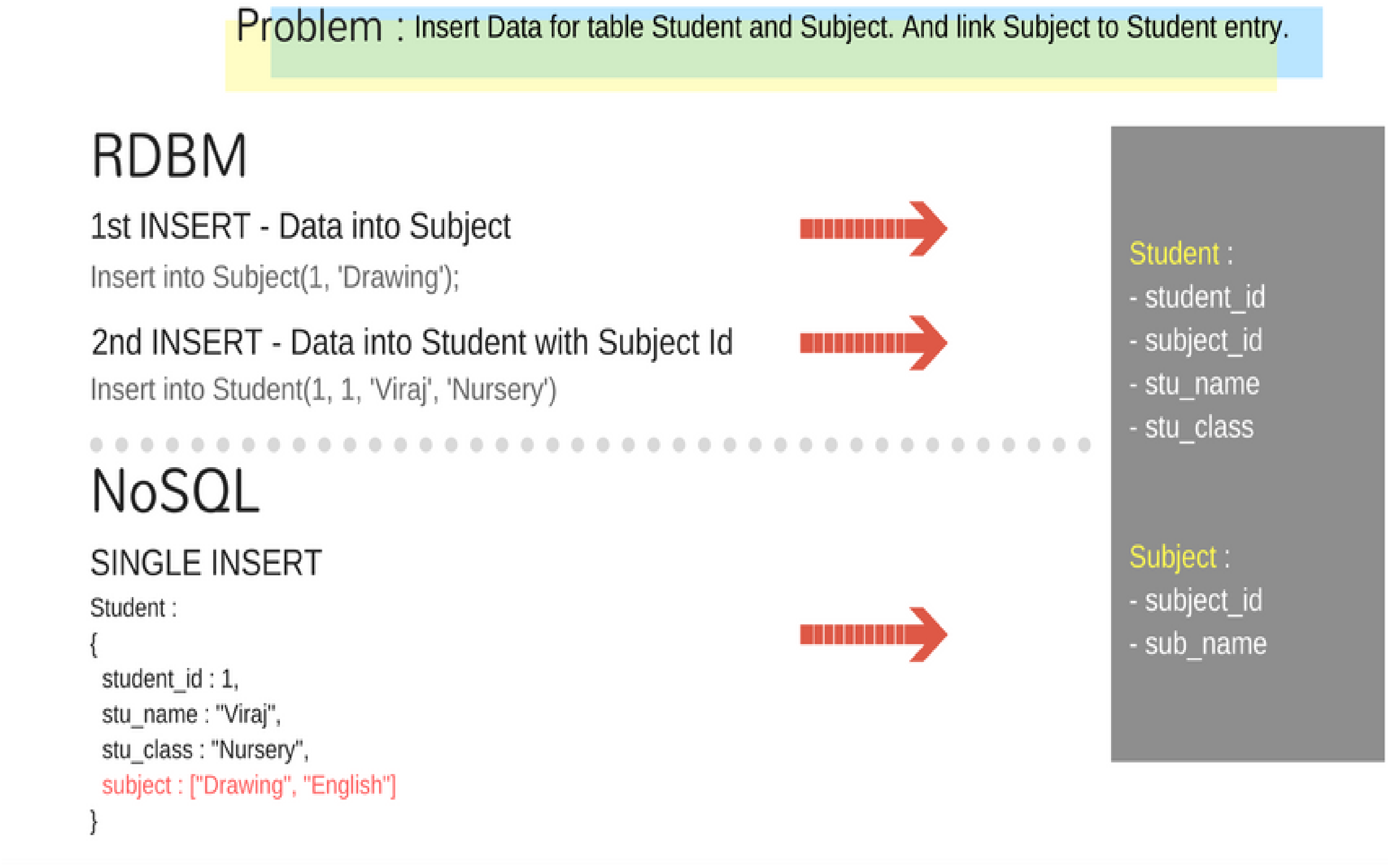
Key Features of MongoDB

- MongoDB provides high performance. Input/Output operations are lesser than relational databases due to support of embedded documents(data models) and Select queries are also faster as Indexes in MongoDB supports faster queries.

Key Features of MongoDB

- MongoDB has a rich Query Language, supporting all the major CRUD operations. The Query Language also provides good Text Search and Aggregation features.
- **Auto Replication** feature of MongoDB leads to High Availability. It provides an automatic failover mechanism, as data is restored through backup(replica) copy if server fails.
- Sharding is a major feature of MongoDB. Horizontal Scalability is possible due to sharding.
- MongoDB supports multiple Storage Engines. When we save data in form of documents(NoSQL) or tables(RDBMS) who saves the data? It's the Storage Engine. Storage Engines manages how data is saved in memory and on disk.
-

Key Features of MongoDB



Problem : Insert Data for table Student and Subject. And link Subject to Student entry.

RDBM

1st INSERT - Data into Subject

Insert into Subject(1, 'Drawing');

2nd INSERT - Data into Student with Subject Id

Insert into Student(1, 1, 'Viraj', 'Nursery')

NoSQL

SINGLE INSERT

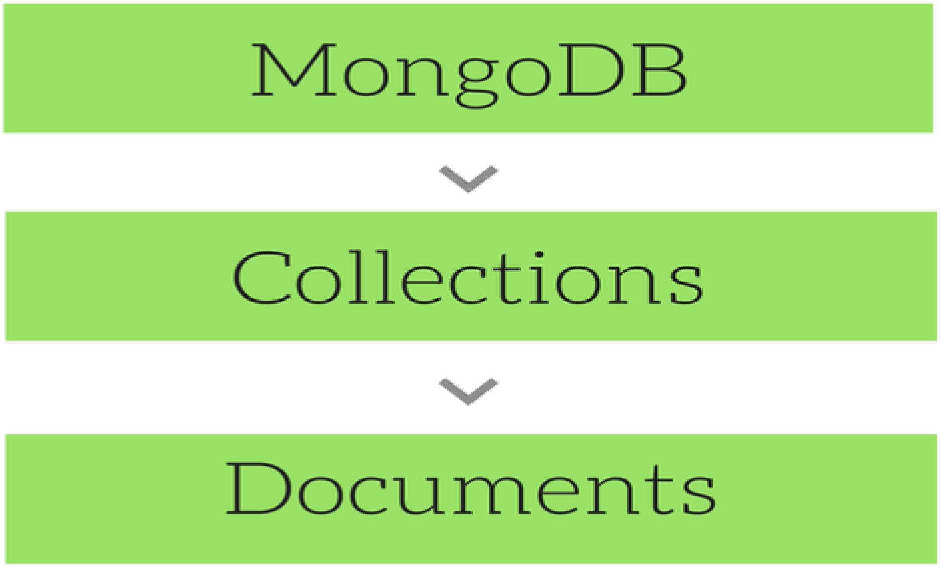
```
Student :
{
  student_id : 1,
  stu_name : "Viraj",
  stu_class : "Nursery",
  subject : ["Drawing", "English"]
}
```

Student :
- student_id
- subject_id
- stu_name
- stu_class

Subject :
- subject_id
- sub_name

Key Features of MongoDB

- **Overview of MongoDB**
- MongoDB consists of a set of databases. Each database again consists of Collections. Data in MongoDB is stored in collections. The below figure depicts the typical database structure in MongoDB.



MongoDB

Collections

Documents

Key Features of MongoDB

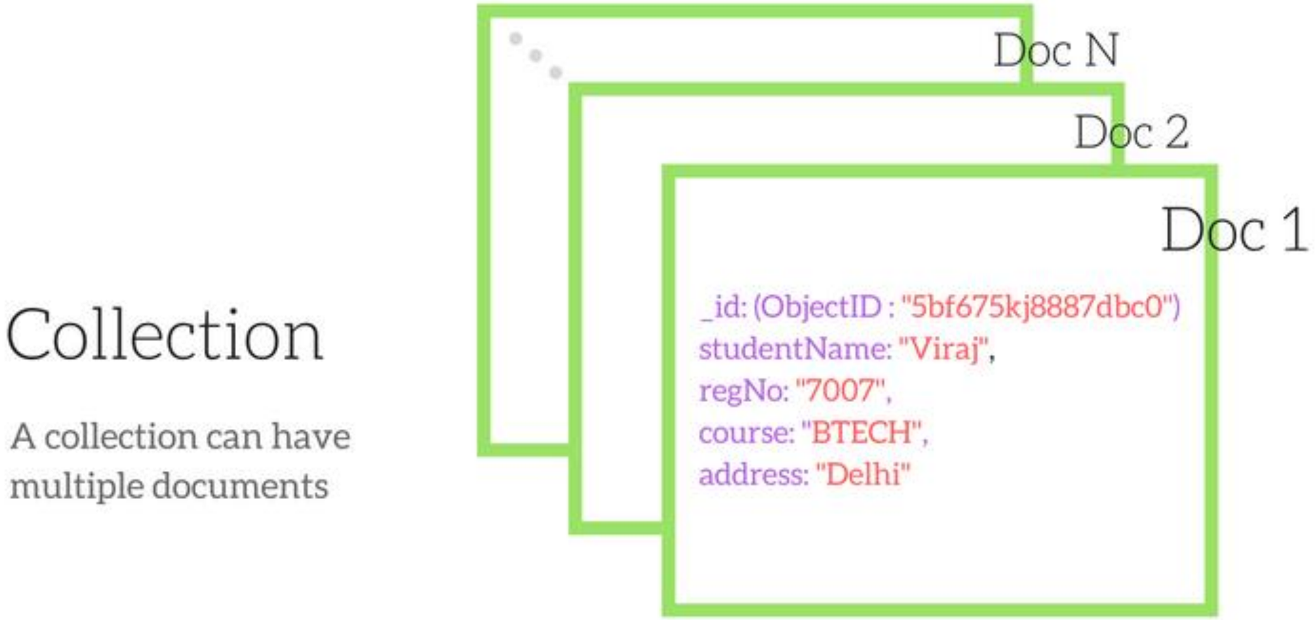
- **Database**
- Database in MongoDB is nothing but a container for collections.

- **Collections**
- Collection is nothing but a set of MongoDB documents. These documents are equivalent to the row of data in tables in RDBMS.

Documents

- Document in MongoDB is nothing but the set of key-value pairs.
- Since MongoDB is considered as a schema-less database, each collection can hold different type of objects. Every object in a collection is known as Document, which is represented in a JSON like (JavaScript Object Notation) structure(nothing but a list of key-value pair).

Sample Data in MongoDB



A collection can have multiple documents

```
_id: (ObjectID : "5bf675kj8887dbc0")
studentName: "Viraj",
regNo: "7007",
course: "BTECH",
address: "Delhi"
```

MongoDB vs SQL Databases

- It is a well known fact that SQL databases have ruled the world of data technologies and have been the primary source of data storage for over 4 decades. Generally the SQL databases are used, mainly for accessing relational databases.
- **Oracle** and **Microsoft SQL Server** ruled the segment, but as the Web development market paced up, there came a shift towards usage of open source databases like **MySQL**, **Postgres** etc. But RDBMS was still the first choice.

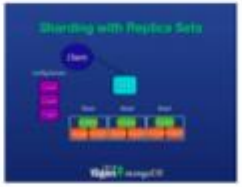
MongoDB vs SQL Databases

SQL Database	**NoSQL Database (MongoDB)**
Relational database	Non-relational database
Supports SQL query language	Supports JSON query language
Table based	Collection based and key-value pair
Row based	Document based
Column based	Field based
Support foreign key	No support for foreign key
Support for triggers	No Support for triggers
Contains schema which is predefined	Contains dynamic schema

Advantages of MongoDB

Faster process

Open Source



Sharding

mongoDB

Schemaless

```
_id: {ObjectID : "[illegible]"},
studentName: "[illegible]",
regNo: "[illegible]"
course: "MCA",
address: "Bangalore"
```

Document based

No SQL Injection

Creating a Database

- Open the command prompt and navigate to the /**bin** folder of the MongoDB using the cd command and execute the command mongod there. This will initiate the MongoDB server. We have to keep this command prompt window alive, as this is running MongoDB. To stop the MongoDB server, simply enterexit and press Enter.
- Now, Open another command prompt and navigate to the /**bin** folder of the MongoDB again and execute the command mongo. This will open up the client to run the MongoDB commands.

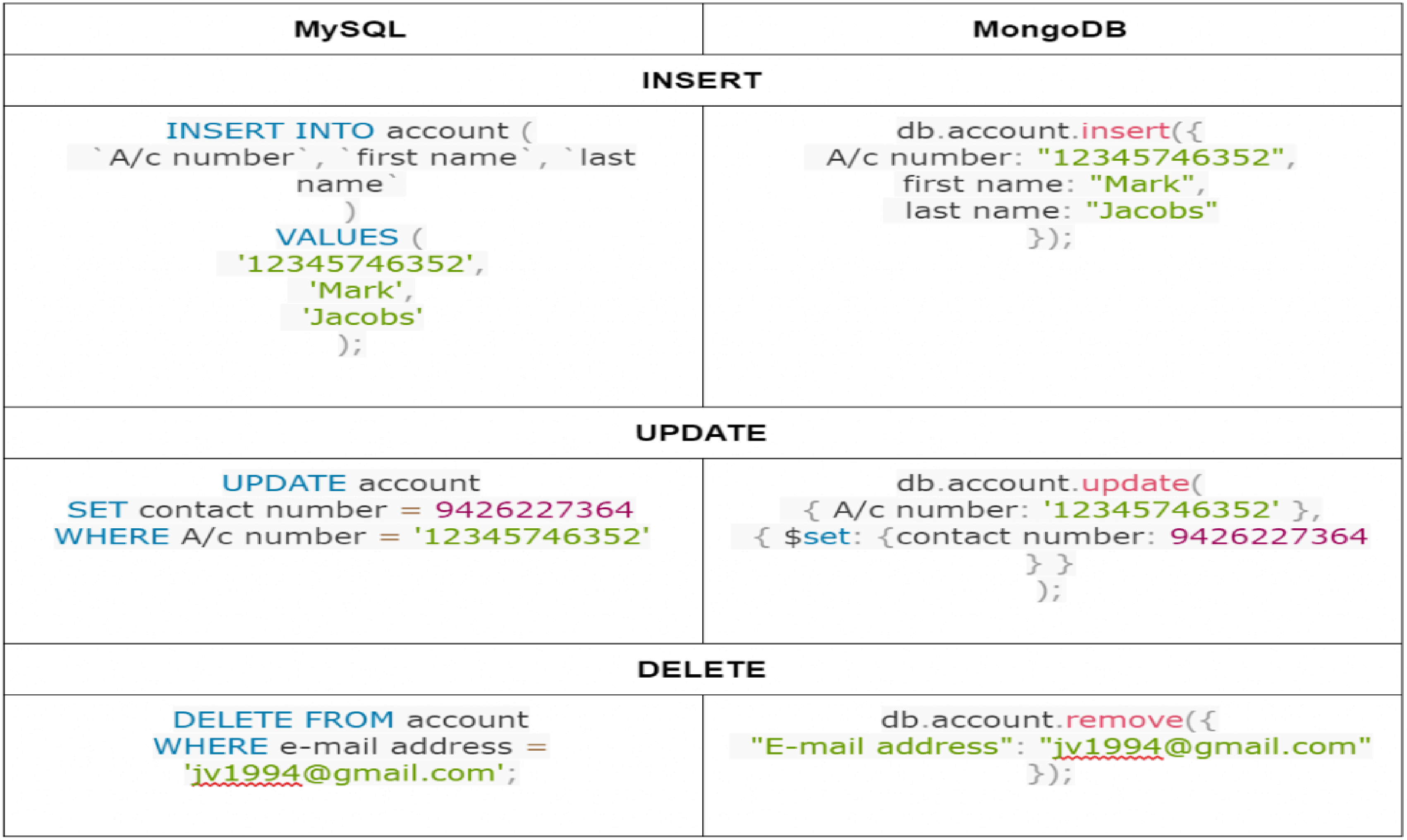
Creating a Database

- **use *database_name***
- To check the current connected database, Use Command
  - **db**
- To see the list of all the databases in MongoDB, use command
  - **show dbs**
- Please note that the newly created dstabase **mynewdatabase** has not been listed after running the above command. This is because, no records have been inserted into that database yet. Just insert one record and then run the command again as shown below:

Creating a Database

- To Insert data, run the following command.
- db.student.insert({ **name** : "*Viraj*" })**NOTE :** In MongoDB, test will be the default database. If no database is created, then all the data will be stored in the test database.
- **Drop a Database**
- First check the list of databases available as shown below, using the show dbs command.

MongoDB CRUD Operations



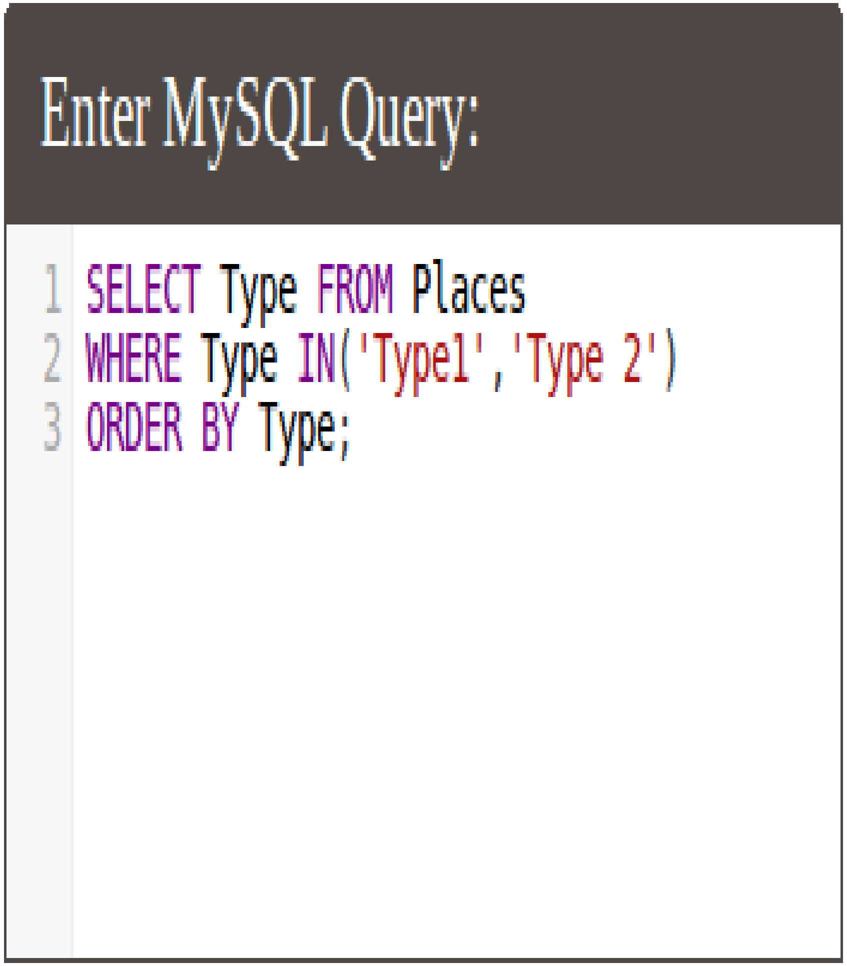
MySQL	MongoDB
INSERT	
INSERT INTO account (`A/c number`, `first name`, `last name`) VALUES ('12345746352', 'Mark', 'Jacobs');	db.account.insert({ A/c number: "12345746352", first name: "Mark", last name: "Jacobs" });
UPDATE	

UPDATE account
SET contact number = 9426227364
WHERE A/c number = '12345746352'
	db.account.update(
	{ A/c number: '12345746352' },
	{ $set: {contact number: 9426227364
	} }
	);

DELETE	

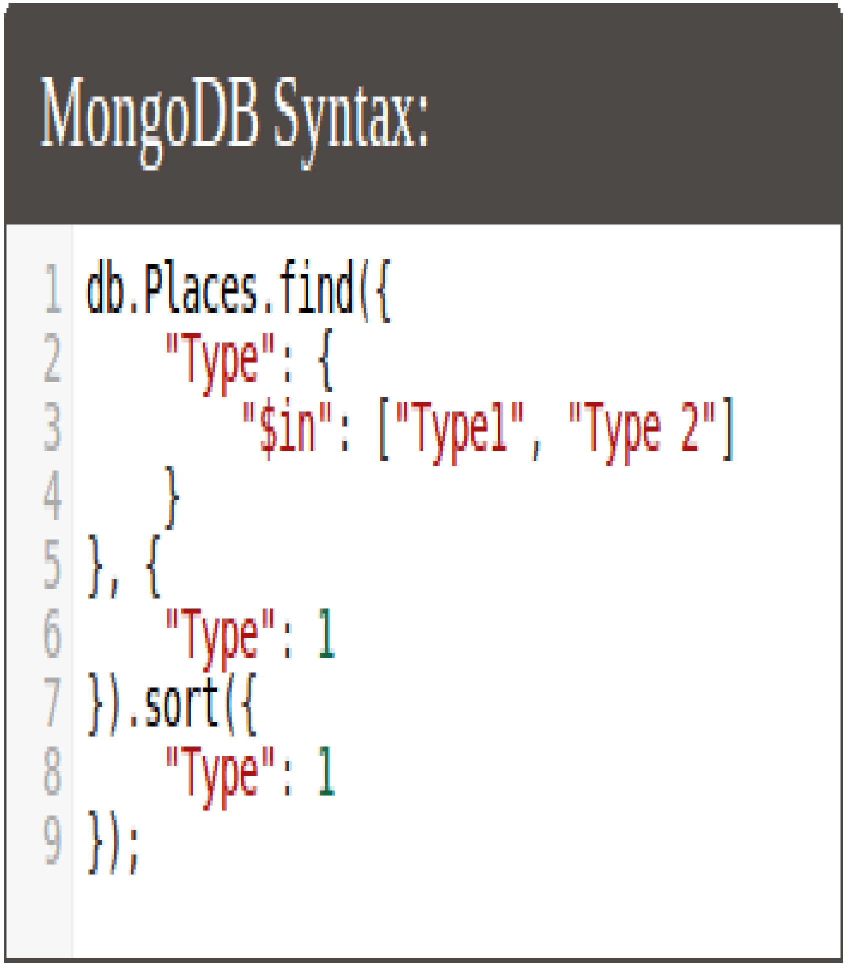
DELETE FROM account
WHERE e-mail address =
'jv1994@gmail.com';
	db.account.remove({
	"E-mail address": "jv1994@gmail.com"
	});

MongoDB CRUD Operations



Enter MySQL Query:

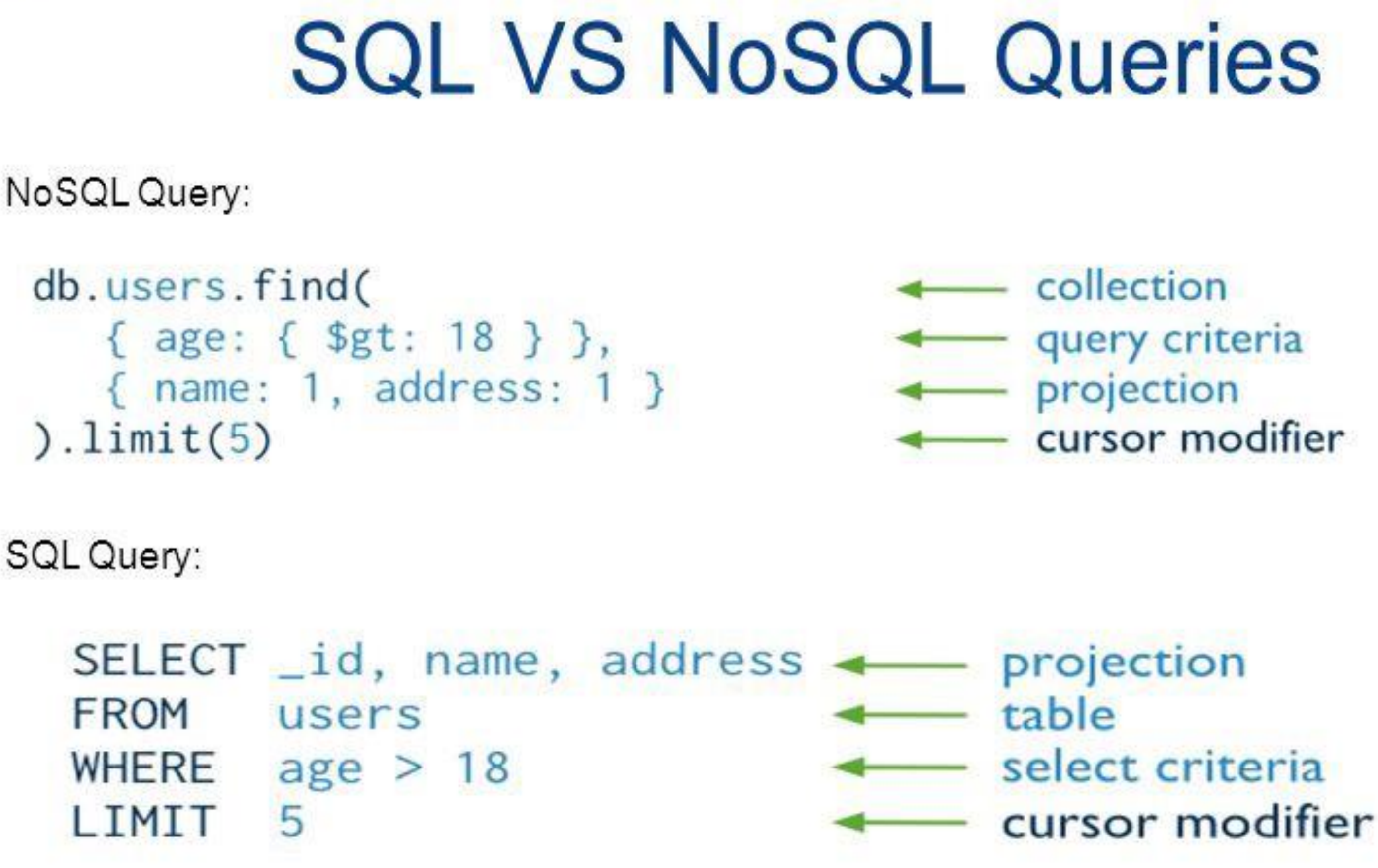
```
SELECT Type FROM Places
WHERE Type IN('Type1','Type 2')
ORDER BY Type;
```



MongoDB Syntax:

```
db.Places.find({
    "Type": {
        "$in": ["Type1", "Type 2"]
    }
}, {
    "Type": 1
}).sort({
    "Type": 1
});
```

MongoDB CRUD Operations



SQL VS NoSQL Queries

NoSQL Query:

```
db.users.find(                          ←  collection
   { age: { $gt: 18 } },                ←  query criteria
   { name: 1, address: 1 }              ←  projection
).limit(5)                              ←  cursor modifier
```

SQL Query:

```
SELECT _id, name, address               ←  projection
FROM   users                            ←  table
WHERE  age > 18                         ←  select criteria
LIMIT  5                                ←  cursor modifier
```

Delete Only One Document that Matches a Condition

To delete at most a single document that matches a specified filter (even though multiple documents may match the specified filter)
use the `db.collection.deleteOne()` method.
The following example deletes the *first* document where `status` is `"D"`:

```
db.inventory.deleteOne( { status: "D" } )
```

Delete All Documents

To delete all documents from a collection, pass an empty filter document {} to the db.collection.deleteMany() method.

The following example deletes *all* documents from the inventory collection:

```
db.inventory.deleteMany({})
```

MONGODB INDEX

Update a Single Document

The following example uses the `db.collection.updateOne()` method on the `inventory` collection to update the *first* document where `item` equals `"paper"`:

```
db.testcollection.updateOne(  { "Name": "Komal" },  {    $set: { "Name":"Ritu Singh" }   })
```

Rename the Field :

```
db.collectionname.update({_id:1},{$rename:{'location' :'address'}})
```

MONGODB UPDATE

```
db.testcollection.updateMany(   { "Name": "Komal" },   {     $set: {
"Name":"Ritu Singh" }   })
```

MONGODB UPDATE

The following example uses the db.collection.updateMany() method on the inventory collection to update all documents where qty is less than 50:

MONGODB INDEX

```
db.inventory.updateMany(

 { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" }

   }
)
```

MONGODB _id

The _id field is always the first field in the document.
In MongoDB, each document stored in a collection requires a unique _id field that acts as a primary key. If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field.

By default, MongoDB creates a unique index on the _id field during the creation of a collection.

The _id field is always the first field in the documents.

Serializing and de-serializing data

In the context of data storage, **serialization** is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later

JSON is a format that encodes objects in a string. **Serialization means to convert an object into that string**, and **deserialization is its inverse operation (convert string -> object)**.

The opposite operation, extracting a data structure from a series of bytes, is **deserialization**

This process of serializing an object is also called marshalling an object in some situations. The opposite operation, extracting a data structure from a series of bytes, is **deserialization**,

(also called **unserialization** or **unmarshalling**)

Serializing and de-serializing data

EXAMPLE

Say, you have an object:
{foo: [1, 4, 7, 10], bar: "baz"}

serializing into JSON will convert it into a string:
'{"foo":[1,4,7,10],"bar":"baz"}'

which can be stored or sent through wire to anywhere. The receiver can then deserialize this string to get back the original object.
{foo: [1, 4, 7, 10], bar: "baz"}.

JSON AND BSON

BSON is just binary **JSON** (a superset of **JSON** with some more data types, most importantly binary byte array). It is a serialization format used in MongoDB.

JSON data contains its data basic in **JSON** format.
BSON gives extra datatypes over the **JSON** data.

Since its initial formulation, BSON has been extended to add some optional non-JSON-native data types, like dates and binary data, without which MongoDB would have been missing some valuable support.

Does MongoDB use BSON, or JSON?

MongoDB stores data in BSON format both internally, and over the network, but that doesn't mean you can't think of MongoDB as a JSON database. Anything you can represent in JSON can be natively stored in MongoDB, and retrieved just as easily in JSON.

Data Support :

Json :

String, Boolean, Number, Array

BSON

String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary

MONGODB SHELL

The MongoDB Shell, mongosh, is a fully functional JavaScript environment for interacting with MongoDB deployments. You can use the MongoDB Shell to test queries and operations directly with your database

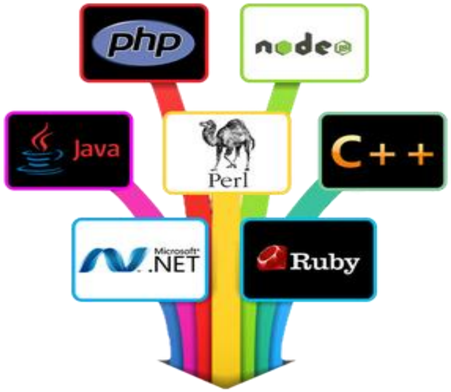
Example : Mongodb Compass , Studio 3t , Robo 3t etc.

MONGODB with Other Language

MongoDB is an open-source database, which is reliable, globally scalable and inexpensive to operate. **MongoDB** provides native drivers for all popular programming languages and frameworks to make development natural.

A list of supported drivers include:

Java, .NET, Ruby, PHP, JavaScript, node.js, Python, Perl, PHP, Scala etc.



MONGODB with Other Language

PHPMoAdmin is MongoDB Administration tool for PHP built on a stripped down version of the Vork high-performance framework. It can help you discover the source of connection issues between PHP and MongoDB.

Mongobee is a java tool which can help you manage changes in your MongoDB database and keep them synchronized with your java application. It provides new approach for adding changes based on Java classes and methods with appropriate annotations.

PyMongo is a Python distribution containing tools for working with MongoDB.

MongoVUE is a .NET Graphical User Interface that gives an elegant and highly usable interface for working with MongoDB.

Node.js Driver is the officially supported node.js driver for MongoDB. It is written in pure JavaScript and provides a native asynchronous Node.js interface to MongoDB. MongoDB also supports some Third-party GUI tools like: Robomongo, Fang of Mongo,

MONGODB with Other Language

Robomongo is a shell-centric cross-platform open source MongoDB management tool. It embeds the same JavaScript engine that powers MongoDB's mongo shells.

Fang of Mongo is a web-based User Interface built with Django and jQuery.
Mongo is a cross-platfrom Management-GUI, implemented in java.

WHAT IS AN INDEX?

An **index** is a performance-tuning method of allowing faster retrieval of records.

Index in database helps to trace the information faster just like an index in a book.

A **database index** is a data structure that improves the speed of data retrieval operations on a database Table (Collection)

Indexes can be created using one or more columns of a database table (Collection).

MONGODB INDEX

Indexes are very important in any database, and with MongoDB it's no different. With the use of Indexes, performing queries in MongoDB becomes more efficient.

If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents that had to be searched in the collection.

MONGODB INDEX

Indexes are special data sets which store a partial part of the collection's data. Since the data is partial, it becomes easier to read this data. This partial set stores the value of a specific field or a set of fields ordered by the value of the field.

MONGODB INDEX

Indexes are good for queries, but having too many indexes can slow down other operations such as the Insert, Delete and Update operation.

If there are frequent insert, delete and update operations carried out on documents, then the indexes would need to change that often, which would just be an overhead for the collection.

MONGODB INDEX

An index can either be based on just one field in the collection, or it can be based on multiple fields in the collection.

How to Create Indexes: createIndex()

The **createIndex** method is used to create an index based on the "Employeeid" of the document.
The '1' parameter indicates that when the index is created with the "Employeeid" Field values, they should be sorted in ascending order. Please note that this is different from the _id field (The id field is used to uniquely identify each document in the collection) which is created automatically in the collection by MongoDB. The documents will now be sorted as per the Employeeid and not the _id field.

How to Create Indexes: createIndex()

Code Explanation:
The createIndex method now takes into account multiple Field values which will now cause the index to be created based on the "Employeeid" and "EmployeeName". The Employeeid:1 and EmployeeName:1 indicates that the index should be created on these 2 field values with the :1 indicating that it should be in ascending order.

How to Create Indexes: createIndex()

```
db.Employee.getIndexes()
```

The output returns a document which just shows that there are 2 indexes in the collection which is the _id field, and the other is the Employee id field. The :1 indicates that the field values in the index are created in ascending order.

Types Of Index In MongoDB

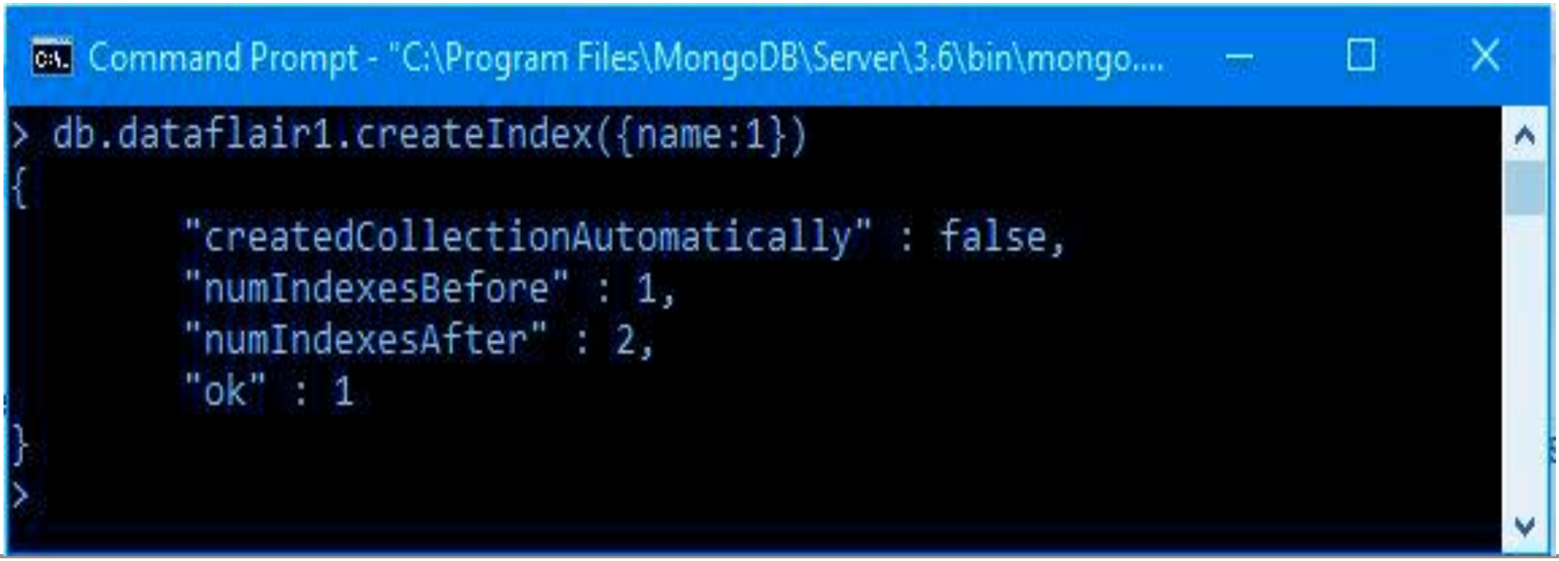
Default Index

In MongoDB indexing, all the collections have a default index on the _id field. If we don't specify any value for the _id the MongoDB will create _id field with an object value. This index prevents clients from creating two documents with the same value _id field.

Types Of Index In MongoDB

Single Field Index

MongoDB supports user-defined indexes like single field index. A single field index is used to create an index on the single field of a document.



Command Prompt - "C:\Program Files\MongoDB\Server\3.6\bin\mongo....

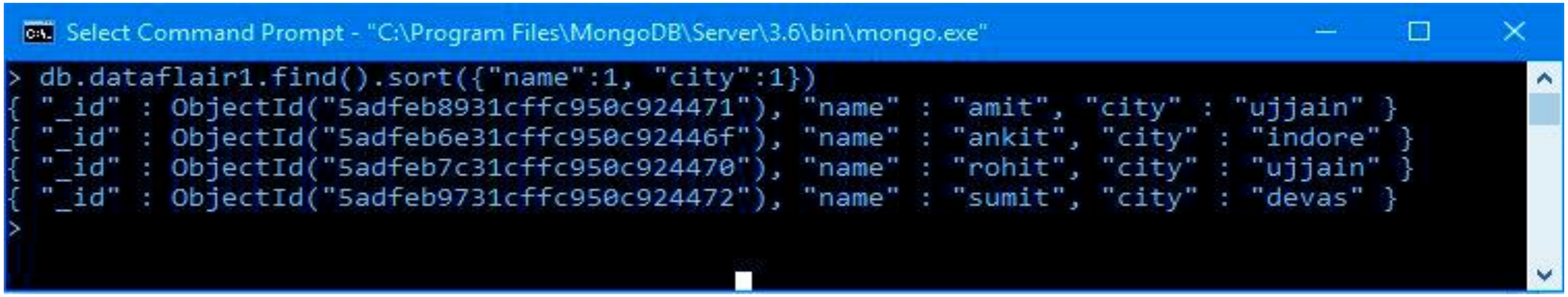
```
> db.dataflair1.createIndex({name:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Types Of Index In MongoDB

Compound Index

MongoDB supports a user-defined index on multiple fields as well. For this MongoDB has a compound index. There sequential order of fields for a compound index.

db.dataflair1.**find**().**sort**({“name”:1,”city”:1})



Select Command Prompt - "C:\Program Files\MongoDB\Server\3.6\bin\mongo.exe"

```
> db.dataflair1.find().sort({"name":1, "city":1})
{ "_id" : ObjectId("5adfeb8931cffc950c924471"), "name" : "amit", "city" : "ujjain" }
{ "_id" : ObjectId("5adfeb6e31cffc950c92446f"), "name" : "ankit", "city" : "indore" }
{ "_id" : ObjectId("5adfeb7c31cffc950c924470"), "name" : "rohit", "city" : "ujjain" }
{ "_id" : ObjectId("5adfeb9731cffc950c924472"), "name" : "sumit", "city" : "devas" }
>
```

INDEX LIMITATIONS:

Maximum Ranges

A collection cannot have more than 64 indexes.

The length of the index name cannot be longer than 125 characters.

A compound index can have maximum 31 fields indexed.

Index

- Defining indexes are important for faster and efficient searching of documents in a collection.
- Indexes can be created by using the createIndex method. Indexes can be created on just one field or multiple field values.
- Indexes can be found by using the getIndexes method.
- Indexes can be removed by using the dropIndex for single indexes or dropIndexes for dropping all indexes.

FULL TEXT SEARCH IN MONGODB

Using MongoDB full-text search, you can define a text index on any field in the document whose value is a string or an array of strings. When we create a text index on a field, MongoDB tokenizes and stems the indexed field's text content, and sets up the indexes accordingly.

The $text operator assigns a score to each document that contains the search term in the indexed fields. The score represents the relevance of a document to a given text search query.

Full-text Search Basics

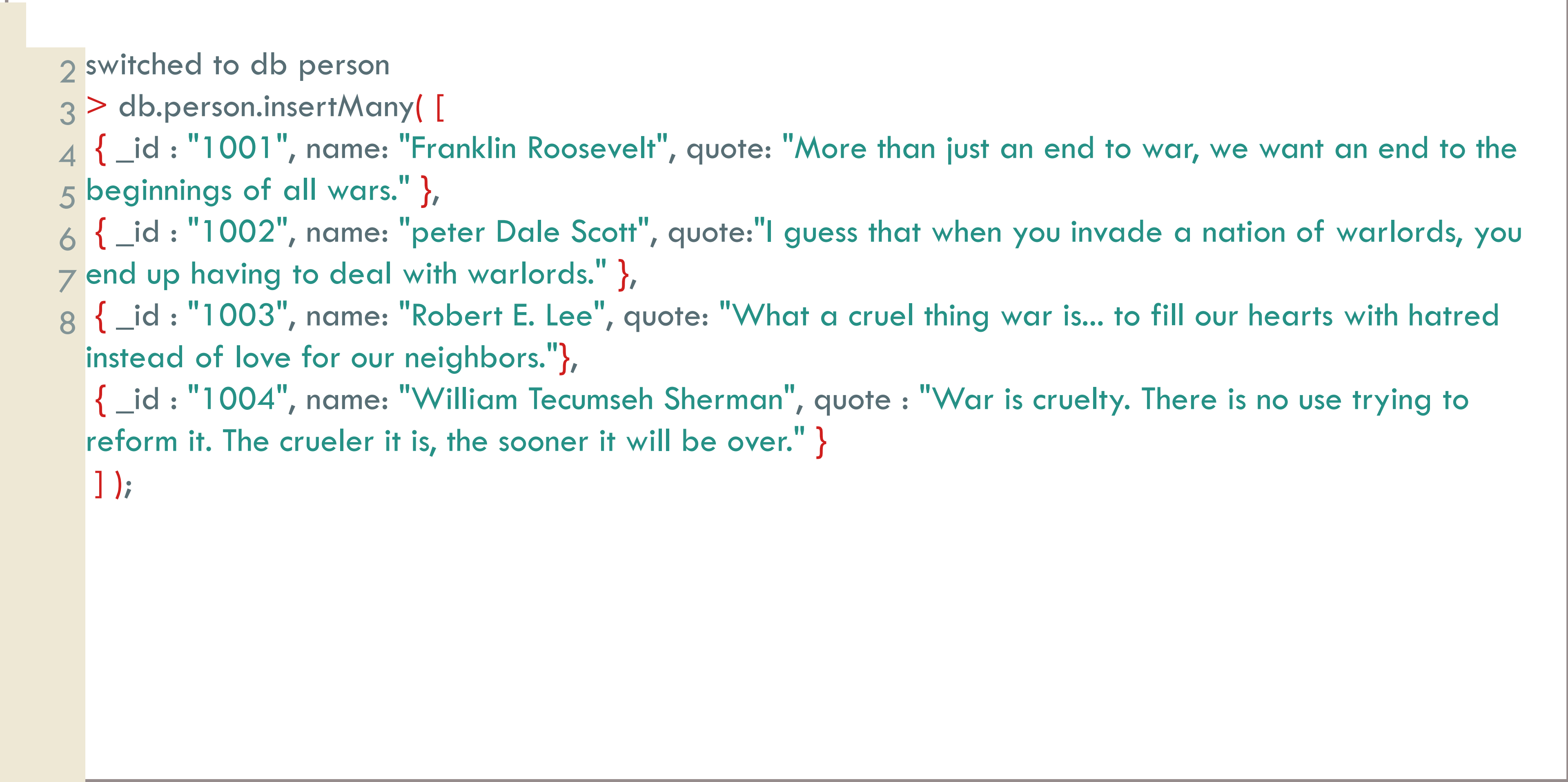
A good way to understand the concept of full-text search is to think about a typical Google search. When we use Google, we find content by providing a series of text, strings, phrases or keywords; in return, a number of results will be returned. In MongoDB, full-text search allows you to perform complex queries that are similar to those you'd perform using a search engine. You can search for phrases and stemmed variations on a word, and it's also possible to exclude certain "negated" terms from your results.

Here are a couple of common scenarios where full-text search plays a key role:

- searching for a certain topic on the web, whether we search Wiki or Google
- searching for a name or term within social networks

Full-text Search Basics : Example

The first step is to connect to your MongoDB server and perform the following commands:



```
switched to db person
> db.person.insertMany( [
 { _id : "1001", name: "Franklin Roosevelt", quote: "More than just an end to war, we want an end to the
beginnings of all wars." },
 { _id : "1002", name: "peter Dale Scott", quote:"I guess that when you invade a nation of warlords, you
end up having to deal with warlords." },
 { _id : "1003", name: "Robert E. Lee", quote: "What a cruel thing war is... to fill our hearts with hatred
instead of love for our neighbors."},
 { _id : "1004", name: "William Tecumseh Sherman", quote : "War is cruelty. There is no use trying to
reform it. The crueler it is, the sooner it will be over." }
 ] );
```

Create Sample Dataset

To verify if the insert operation was a success, use the following command: “

```
db.person.find().pretty();
```

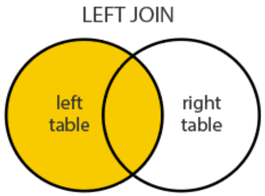
Create Text Index in MongoDB

This functionality is provided by MongoDB to support text search on strings within MongoDB collections or documents.

db.person.createIndex({"quote":"text"})

$lookup (aggregation)

- ❑ Performs a left outer join to an unsharded collection in the *same* database to filter in documents from the “joined” collection for processing.
- ❑ **Left Outer Join:** Returns all the rows from the **LEFT** table and matching records between both the tables.



- ❑ To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the “joined” collection. The $lookup stage passes these reshaped documents to the next stage.

$lookup (aggregation)

The operation would correspond to the following pseudo-SQL statement:

```
SELECT *, inventory_docs
FROM orders
WHERE inventory_docs IN (SELECT *
FROM inventory
WHERE sku= orders.item);
```

$lookup (aggregation)

Equality Match

To perform an equality match between a field from the input documents with a field from the documents of the “joined” collection, the $lookup stage has the following syntax:

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

$lookup : Example

Perform a Single Equality Join with $lookup

Create a collection orders with the following documents:

```
db.orders.insert([
   { "_id" : 1, "item" : "almonds", "price" : 12, "quantity" : 2 },
   { "_id" : 2, "item" : "pecans", "price" : 20, "quantity" : 1 },
   { "_id" : 3  }
])
```

Create another collection inventory with the following documents:

```
db.inventory.insert([
   { "_id" : 1, "sku" : "almonds", "description": "product 1", "instock" : 120 },
   { "_id" : 2, "sku" : "bread", "description": "product 2", "instock" : 80 },
   { "_id" : 3, "sku" : "cashews", "description": "product 3", "instock" : 60 },
   { "_id" : 4, "sku" : "pecans", "description": "product 4", "instock" : 70 },
   { "_id" : 5, "sku": null, "description": "Incomplete" },
   { "_id" : 6 }
])
```

$lookup : Example

The following aggregation operation on the orders collection joins the documents from orders with the documents from the inventory collection using the fields item from the orders collection and the sku field from the inventory collection:

```
db.orders.aggregate([
  {
    $lookup:
      {
        from: "inventory",
        localField: "item",
        foreignField: "sku",
        as: "inventory_docs"
      }
 }
])
```

$lookup (aggregation)

The $lookup takes a document with the following fields

from	Specifies the collection in the *same* database to perform the join with. The from collection cannot be sharded
localField	Specifies the field from the documents input tthe $lookup stage. $lookup performs an equality match on the localField to the foreignField from the documents of the from collection. If an input document does not contain the localField, the $lookup treats the field as having a value of null for matching purposes.

$lookup (aggregation)

The $lookup takes a document with the following fields

foreignField

Specifies the field from the documents in the from collection. $lookup performs an equality match on the foreignField to the localField from the input documents. If a document in the from collection does not contain the foreignField, the $lookup treats the value as null for matching purposes.

as

Specifies the name of the new array field to add to the input documents. The new array field contains the matching documents from the from collection. If the specified name already exists in the input document, the existing field is *overwritten*

$project (aggregation)

- Passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.
- The $project takes a document that can specify the inclusion of fields, the suppression of the _id field, the addition of new fields, and the resetting of the values of existing fields. Alternatively, you may specify the *exclusion* of fields.

$project (aggregation)

The $project specifications have the following forms:

Form	Description
<field>: <1 or true>	Specifies the inclusion of a field.
_id: <0 or false>	Specifies the suppression of the _id field.
<field>:<0 or false>	Specifies the exclusion of a field.

Include Existing Fields
The _id field is, by default, included in the output documents. To include any other fields from the input documents in the output documents, you must explicitly specify the inclusion in $project.

Suppress the _id Field
By default, the _id field is included in the output documents. To exclude the _id field from the output documents, you must explicitly specify the suppression of the _id field in $project.

$project (aggregation)

Examples

Include Specific Fields in Output Documents

Consider a books collection with the following document:

```
db.books.insert(
{
  "_id" : 1,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5
}
)
```

The following $project stage includes only the _id, title, and the author fields in its output documents:

```
db.books.aggregate( [ { $project : { title : 1 , author : 1 } } ] )
```

The operation results in the following document:

```
{ "_id" : 1, "title" : "abc123", "author" : { "last" : "zzz", "first" : "aaa" } }
```

$project (aggregation)

The following $project stage excludes the _id field but includes the title, and the author fields in its output documents:

db.books.aggregate([{ $project : { _id: 0, title : 1 , author : 1 } }])

The operation results in the following document:

{ "title" : "abc123", "author" : { "last" : "zzz", "first" : "aaa" } }

$project (aggregation)

Exclude Fields from Embedded Documents

Consider a books collection with the following document:

```
db.books.insert(
{
  "_id" : 1,
  title: "abc123",
  isbn: "0001122223334",
  author: { last: "zzz", first: "aaa" },
  copies: 5,
  lastModified: "2016-07-28"
}
)
```

The following $project stage excludes the author.first and lastModified fields from the output:

`db.books.aggregate( [ { $project : { "author.first" : 0, "lastModified" : 0 } } ] )`

$project (aggregation)

Specifications result :

```
{
   "_id" : 1,
   "title" : "abc123",
   "isbn" : "0001122223334",
   "author" : {
      "last" : "zzz"
   },
   "copies" : 5,
}
```

What is Cursor in MongoDB?

When the **db.collection.find ()** function is used to search for documents in the collection, the result returns a pointer to the collection of documents returned which is called a cursor.
By default, the cursor will be iterated automatically when the result of the query is returned. But one can also explicitly go through the items returned in the cursor one by one. If you see the below example, if we have 3 documents in our collection, the cursor object will point to the first document and then iterate through all of the documents of the collection.

example shows how this can be done.

```
var myEmployee = db.Employee.find( { Employeeid : { $gt:2 }});

          while(myEmployee.hasNext())

          {
                    print(tojson(myEmployee.next()));
          }
```

Code Explanation:

1.First we take the result set of the query which finds the Employee's whose id is greater than 2 and assign it to the JavaScript variable 'myEmployee'

2 .Next we use the while loop to iterate through all of the documents which are returned as part of the query.

3. Finally for each document, we print the details of that document in JSON readable format.

$explain

It provides information on the query, indexes used in the query and some statistics. It is good to use this if you want to know how well your indexes are optimized.

```
{
"_id": ObjectId("53402597d852426020000002"),
"contact": "1234567809",
"dob": "01-01-1991",
"gender": "M",
"name": "ABC",
"user_name": "abcuser"
}
```

$explain

```
db.examples.ensureIndex({gender:1,user_name:1})
```

Now we will use $explain query over it.

```
db.examples.find({gender:"M"},{user_name:1,_id:0}).explain()
db.LVCCollection.find({RegNo:"1"}).explain()
```

After executing the above line we will get the following output:

```
{
"cursor" : "BtreeCursor gender_1_user_name_1",
"isMultiKey" : false,
"n" : 1,
"nscannedObjects" : 0,
"nscanned" : 1,
"nscannedObjectsAllPlans" : 0,
"nscannedAllPlans" : 1,
"scanAndOrder" : false,
"indexOnly" : true,
"nYields" : 0,
"nChunkSkips" : 0,
```

$explain

```
"millis" : 0,
"indexBounds" : {
"gender" : [
[
"M",
"M"
]
],
"user_name" : [
[
{
"$minElement" : 1
},
{
"$maxElement" : 1
}]]}}
```

What is a MongoDB Query?

MongoDB query is used to specify the selection filter using query operators while retrieving the data from the collection by **db.find()** method. We can easily filter the documents using the query object.

To apply the filter on the collection, we can pass the query specifying the condition for the required documents as a parameter to this method, which is an optional parameter for db.find() method.

MONGO QUERY LANGUAGE

MongoDB uses **the MongoDB Query Language** (MQL), designed for easy use by developers. **The** documentation compares MQL and SQL **syntax** for common database operations.
These **queries** are sent to the **MongoDB** server present in the Data Layer. Now, the **MongoDB** server receives the **queries** and passes the received **queries** to the storage engine. **MongoDB** server itself does not directly read or write the data to the files or disk or memory.

MONGO QUERY LANGUAGE

Sort

```
db.userdetails.find({"date_of_join" : "16/10/2010","education":"M.C.A."}).sort({"profession":-1}).pretty()
```

$query & $orderby

```
db.userdetails.find({$query : {"date_of_join" : "16/10/2010","education":"M.C.A."}, $orderby : {"profession":-1}}).pretty();
```

MongoDB Covered Query

The MongoDB covered query is one which uses an **index** and does not have to examine any documents. An index will cover a query if it satisfies the following conditions:

All fields in a query are part of an index.

All fields returned in the results are of the same index.

If we take an example of a **collection** named example which has 2 indexes type and item.

```
db.example.createIndex( { type: 1, item: 1 } )
```

This index will cover queries with operations on type and item fields and then return only item field.

```
db.example.find(
{ type: "game", item:/^c/ },
{ item: 1, _id: 0 }
)
```

Here, to cover the query, the **document** must explicitly specify _id: 0 to exclude the _id field from a result. However, this has changed in version 3.6. Over here an index can cover a query on fields within embedded documents.

SQL TO MONGODB MAPPING CHART

https://docs.mongodb.com/manual/reference/sql-comparison/

MONGO QUERY LANGUAGE

The MongoDB server treats all the query parameters as a single object. Some object such as $query, $orderby etc can be used to get the same result as return simple mongo query language. The $query can be evaluated as the WHERE clause of SQL and $orderby sorts the results as specified order.

MONGODB .NOTATION

If we want to fetch documents from the collection "testtable" which contain the value of "community_name" is "MODERN MUSIC" and "valued_friends_id" which is under the "friends" is "harry" and all the said is under "extra" of an JSON style object, the following mongodb command can be used :

```
> db.testtable.find({"extra.community_name" : "MODERN MUSIC","extra.friends.valued_friends_id":"harry"}).pretty();
```

MONGODB .NOTATION

Display all documents from a collection with the help of find() method –

```
db.demo302.find();
```

Following is the query for field selection using dot notation –

```
db.demo302.find({"details.Subject":"MongoDB"},
{"details.Name":0,"details.Age":0,_id:0,Id:0});
```

This will produce the following output –

```
{ "details" : [ { "Subject" : "MongoDB" } ] }
```

What is replication?

Replication is a way of keeping identical copies of data on multiple servers and it is recommended for all production deployments.

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers.

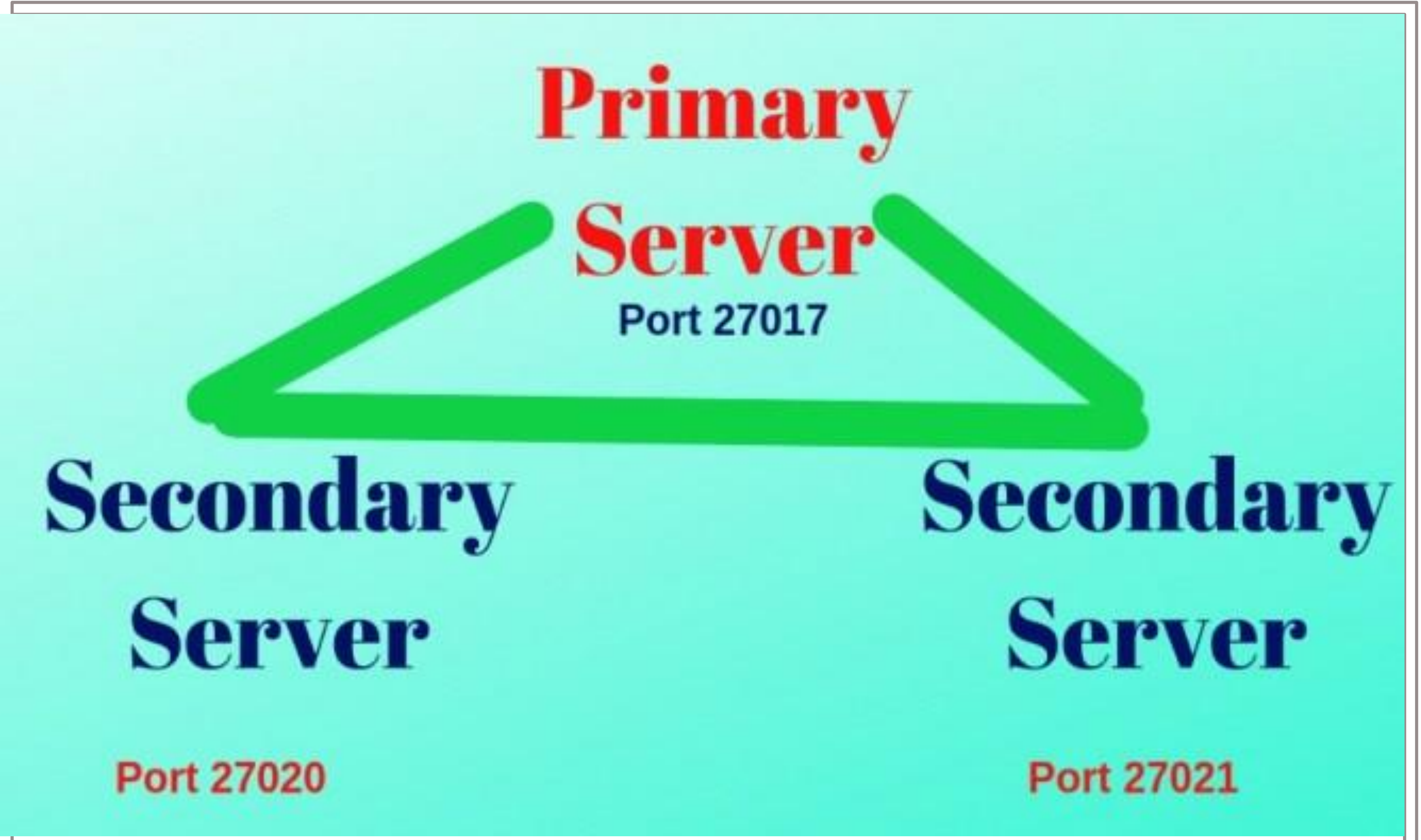
Replication protects a database from the loss of a single server. Replication also allows you to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

Data Replication?

Mirrors the data stored on a server to another server – or servers – instantaneously or nearly instantaneously depending on your infrastructure

Replicated data provides an exact copy of whatever is on your network at a given moment, and is ideal for restoring access quickly to mission-critical data and applications after a disaster

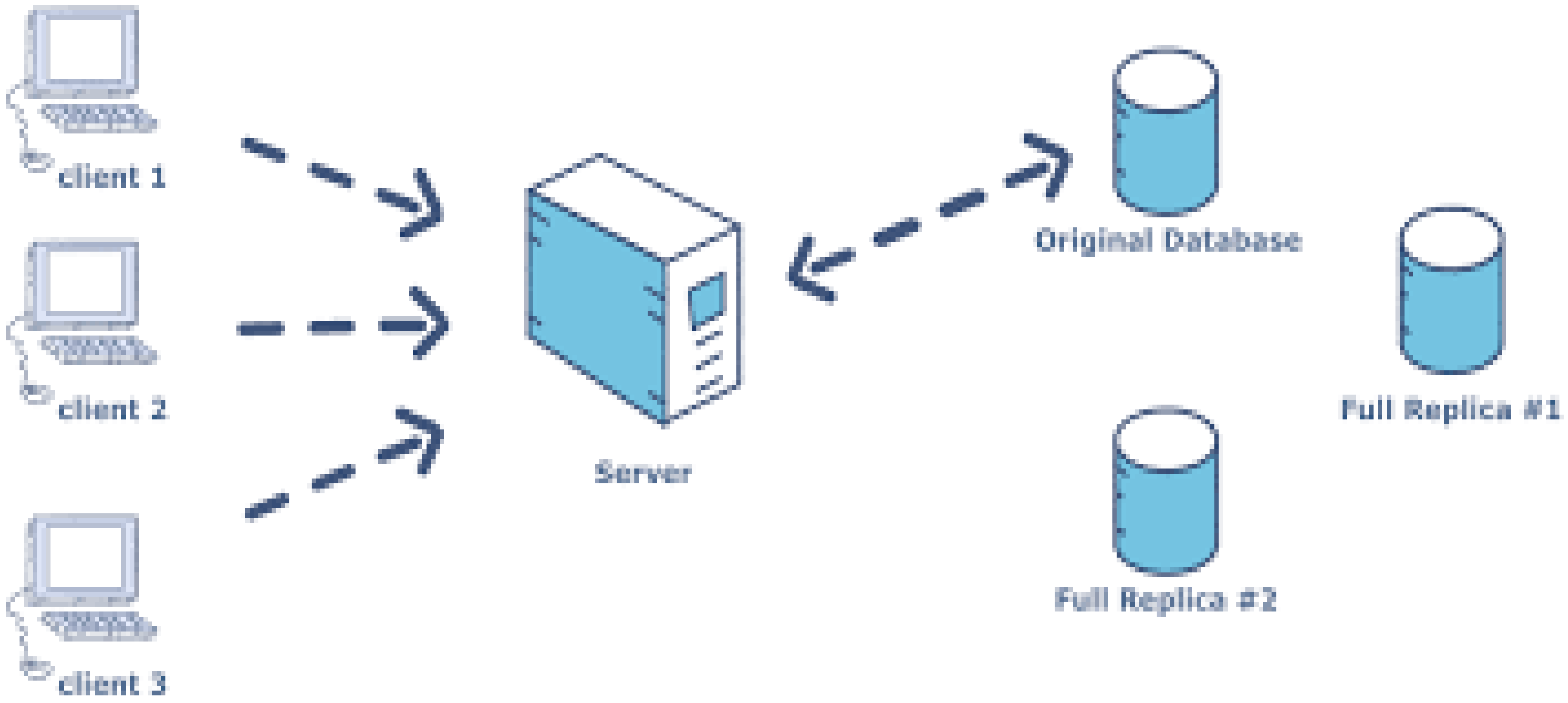
What is replication?



Why Replication?

- To keep your data safe
- High (24*7) availability of data
- Disaster recovery
- No downtime for maintenance (like backups, index rebuilds, compaction)

Why Replication?



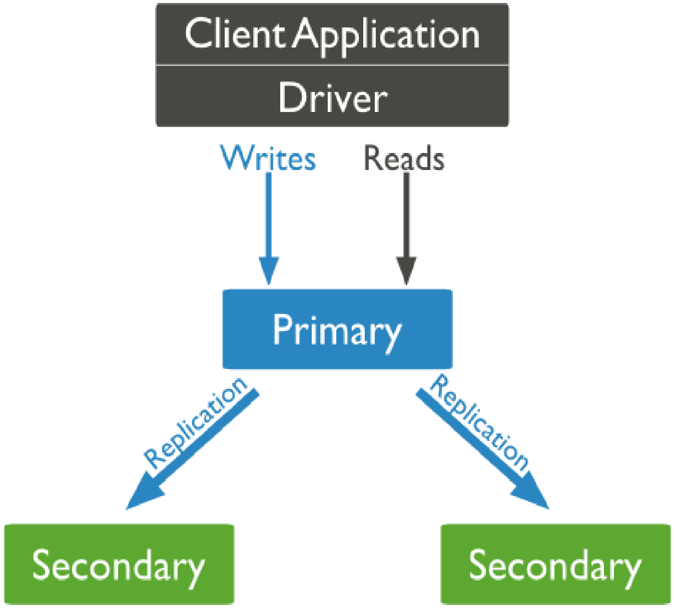
How Replication Works in MongoDB

MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

How Replication Works in MongoDB

A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes

-

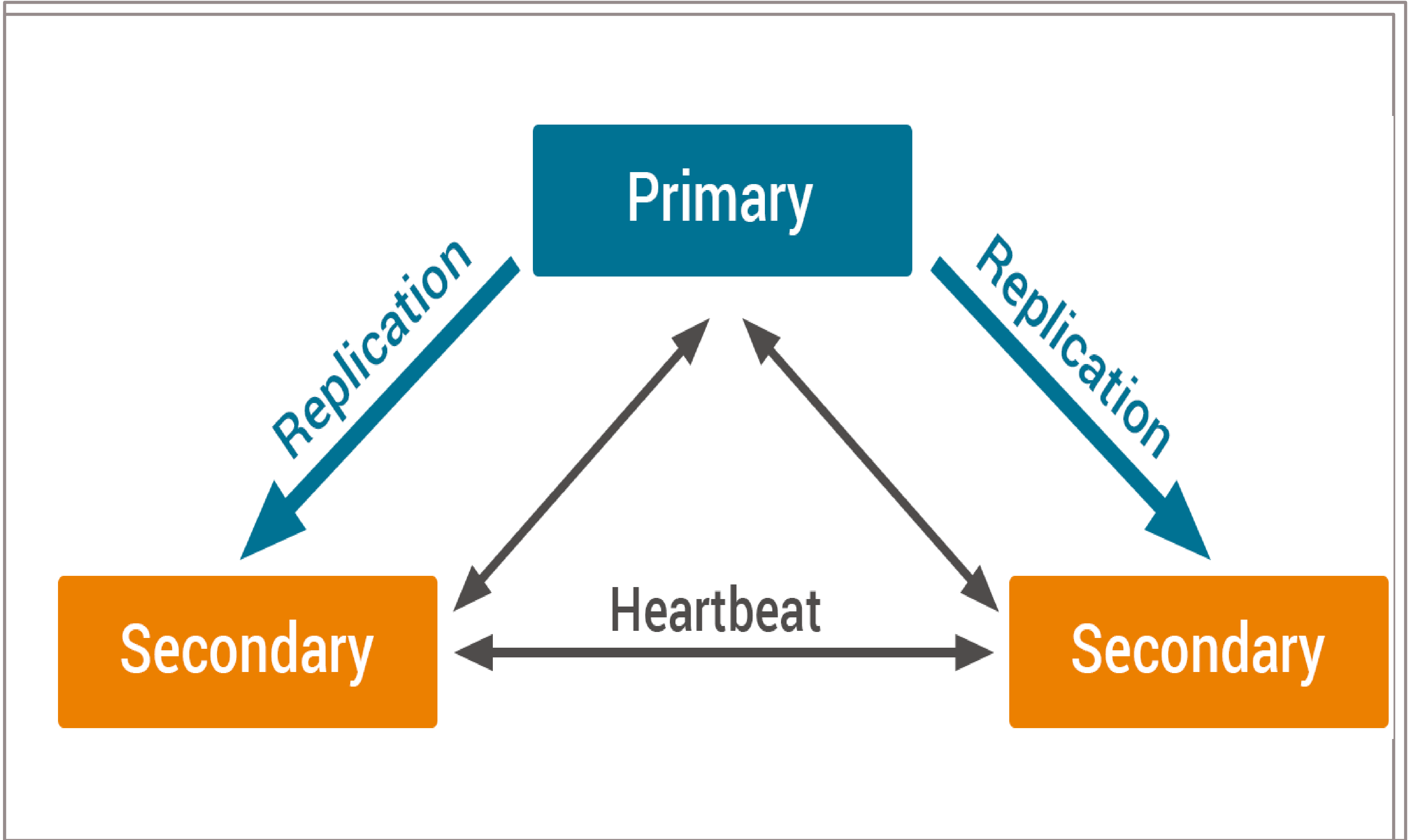


How Replication Works in MongoDB

Replication is referred to the process of ensuring that the same data is available on more than one Mongo DB Server. This is sometimes required for the purpose of increasing data availability.

Because if your main MongoDB Server goes down for any reason, there will be no access to the data. But if you had the data replicated to another server at regular intervals, you will be able to access the data from another server even if the primary server fails.

How Replication Works in MongoDB

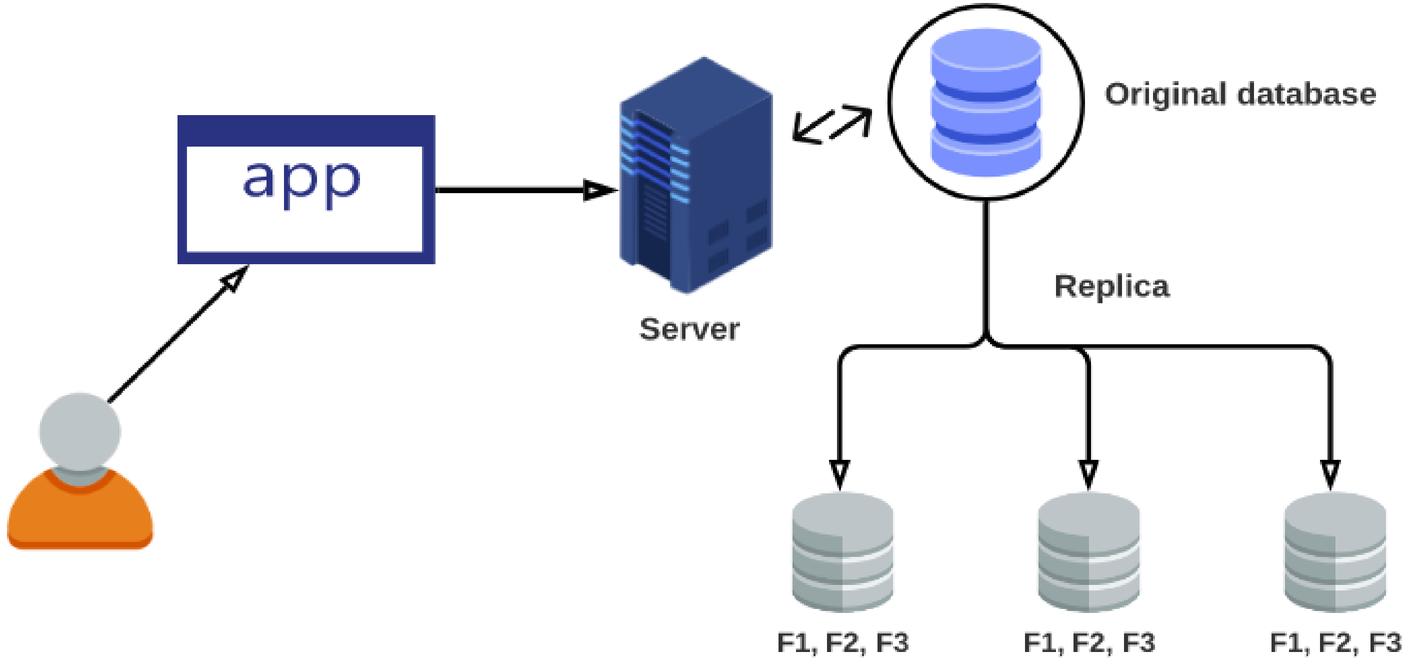


How Replication Works in MongoDB

Another purpose of replication is the possibility of load balancing. If there are many users connecting to the system, instead of having everyone connect to one system, users can be connected to multiple servers so that there is an equal distribution of the load.

How Replication Works in MongoDB

Full replication in DBMS



[FI, F2, F3 are the different fragments of the main database]

Replica Set

- In MongoDB, multiple MongDB Servers are grouped in sets called Replica sets. The Replica set will have a primary server which will accept all the write operation from clients. All other instances added to the set after this will be called the secondary instances which can be used primarily for all read operations.

MongoDB Oplog

- ❑ The MongoDB Oplog origin reads entries from MongoDB Oplog.
- ❑ MongoDB stores information about changes to the database in a local capped collection called an Oplog.

- ❑ The Oplog contains information about changes in data as well as changes in the database.
- ❑ The MongoDB Oplog origin can read any operation written to the Oplog.
- ❑ Use the MongoDB Oplog origin to capture changes in data or the database. To process MongoDB data written to a standard capped or uncapped collection, use the MongoDB origin.

- ❑ The MongoDB Oplog origin includes the CRUD operation type in a record header attribute so generated records can be easily
- ❑ processed by CRUD-enabled destinations.

Mongodb Backup

Creating a backup of an empty database is not very useful, so in this step, we will create an example database and add some data to it.

The easiest way to interact with a MongoDB instance is to use the `mongo` shell. Open it with the `mongo` command.

- `mongo`

Once you have the MongoDB prompt, create a new database called **myDatabase** using the `use` helper.

- `use myDatabase`

Mongodb Backup

All data in a MongoDB database should belong to a *collection*. However, you don’t have to create a collection explicitly. When you use the `insert` method to write to a non-existent collection, the collection is created automatically before the data is written.

You can use the following code to add three small documents to a collection called **myCollection** using the `insert` method:

Mongodb Backup

```
db.myCollection.insert([    {'name':
'Alice', 'age': 30},    {'name':
'Bill', 'age': 25},    {'name':
'Bob', 'age': 35}]);
```

```
db.stats().dataSize;
```

Mongodb Backup

Creating a Backup

To create a backup, you can use a command-line utility called mongodump. By default, mongodump will create a backup of all the databases present in a MongoDB instance. To create a backup of a specific database, you must use the -d option and specify the name of the database. Additionally, to let mongodump know where to store the backup, you must use the -o option and specify a path.

```
mongodump -d myDatabase -o ~/backups/first_backup
```

Restoring the Database

To restore a database using a backup created using `mongodump`, you can use another command line utility called `mongorestore`. Before you use it, exit the `mongo` shell by pressing `CTRL+D`. Using `mongorestore` is very simple. All it needs is the path of the directory containing the backup. Here's how you can restore your database using the backup stored in `~/backupts/first_backup`:

```
mongorestore ~/backups/first_backup/
```

Data Backup

- Copies necessary files and applications to a secondary location such as a hosted or offsite server
- Backups are done periodically throughout the day, and preserve a “snapshot” of your data from a certain point in time
- These snapshots can be used to restore data that has been lost or corrupted on your primary systems
- Because they exist separate from both your network and office, they are protected from anything and everything that can harm your business
- **Backup** involves making a copy or copies of data and storing them offsite in case the original **is** lost or damaged. **Replication is** the act of copying data and then moving data **between** a company's sites, whether those be datacenters, colocation facilities, public, or private clouds.
-

BACKUP VS REPLICATION

Data Replication

- Mirrors the data stored on a server to another server – or servers – instantaneously or nearly instantaneously depending on your infrastructure
- Replicated data provides an exact copy of whatever is on your network at a given moment, and is ideal for restoring access quickly to mission-critical data and applications after a disaster
- The primary focus is on ensuring that you have continuous access to the applications and processes your business needs to function