**Document Orientated Databases**

When relational databases were introduced into the 1970s, data schemas were fairly simple and straightforward, and it made sense to conceive objects as sets of relationships. For example, an article object might be related to a category (an object), a tag (another object), a comment (another object), and so on.

Because relationships between different types of data were specified in the database schema, these relational databases could be queried with a standard Structured Query Language, or SQL. But the environment for data, as well as programming, has changed since the development of the SQL database:

* **The emergence of** [**cloud computing**](http://www.mongodb.com/cloud-computing) has brought deployment and storage costs down dramatically, but only if data can be spread across multiple servers easily without disruption. In a complex SQL database, this is difficult because many queries require multiple large tables to be joined together to provide a response. Executing distributed joins is a very complex problem in relational databases.
* **The need to store unstructured data**, such as social media posts and multimedia, has grown rapidly. SQL databases are extremely efficient at storing structured information, and workarounds or compromises are necessary for storing and querying unstructured data.
* [**Agile development**](http://www.mongodb.com/agile-development) **methods** mean that the database schema needs to change rapidly as demands evolve. SQL databases require their structure to be specified in advance, which means any changes to the information schema require time-consuming ALTER statements to be run on a table.

In response to these changes, new ways of storing data (e.g. [NoSQL databases](http://www.mongodb.com/learn/nosql)) have emerged that allow data to be grouped together more naturally and logically, and that loosen the restrictions on database schema. One of the most popular ways of storing data is a document data model, where each record and its associated data is thought of as a “document”. In a document database, such as MongoDB, everything related to a database object is encapsulated together. Storing data in this way has the following advantages:

* **Documents are independent units** which makes performance better (related data is read contiguously off disk) and makes it easier to distribute data across multiple servers while preserving its locality.
* **Application logic is easier to write.** You don’t have to translate between objects in your application and SQL queries, you can just turn the object model directly into a document.
* **Unstructured data can be stored easily**, since a document contains whatever keys and values the application logic requires. In addition, costly migrations are avoided since the database does not need to know its information schema in advance.

Document databases generally have very powerful query engines and indexing features that make it easy and fast to execute many different optimized queries. The strength of a document database’s query language is an important differentiator between these databases.

## What is NoSQL?

NoSQL encompasses a wide variety of different database technologies that were developed in response to a rise in the volume of data stored about users, objects and products, the frequency in which this data is accessed, and performance and processing needs. Relational databases, on the other hand, were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the cheap storage and processing power available today.

## NoSQL Database Types

* **Document databases** pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
* **Graph stores** are used to store information about networks, such as social connections. Graph stores include Neo4J and HyperGraphDB.
* **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or "key"), together with its value. Examples of key-value stores are Riak and Voldemort. Some key-value stores, such as Redis, allow each value to have a type, such as "integer", which adds functionality.
* **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.

## The Benefits of NoSQL

When compared to relational databases, NoSQL databases are [more scalable and provide superior performance](http://www.mongodb.com/scale), and their data model addresses several issues that the relational model is not designed to address:

* Large volumes of structured, semi-structured, and unstructured data
* Agile sprints, quick iteration, and frequent code pushes
* Object-oriented programming that is easy to use and flexible
* Efficient, scale-out architecture instead of expensive, monolithic architecture

**MongoDB is the Leading NoSQL Document Database**

MongoDB's document data model makes it easy to build on, since it supports unstructured data natively and doesn't require costly and time-consuming migrations when application requirements change. MongoDB's documents are encoded in a JSON-like format, called BSON, which makes storage easy, is a natural fit for modern object-oriented programming methodologies, and is also lightweight, fast and traversable.

In addition, MongoDB supports rich queries and full indexes, distinguishing it from other document databases that make complex queries difficult or require a separate server layer to enable them. Its other features include automatic sharding, replication, and more.

What its great for:

• Websites

• Caching

• High scalability

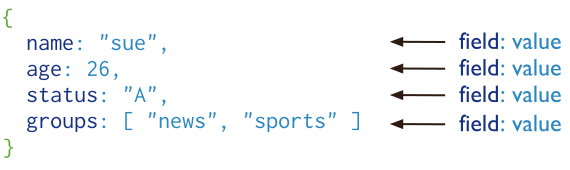
• Storage of program objects and JSON

What its not great for:

• Highly transactional applications

• Problems requiring SQL

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.



The advantages of using documents are:

* Documents (i.e. objects) correspond to native data types in many programming languages.
* Embedded documents and arrays reduce need for expensive joins.
* Dynamic schema supports fluent polymorphism.

### Key Features

#### High Performance

MongoDB provides high performance data persistence. In particular,

* Support for embedded data models reduces I/O activity on database system.
* Indexes support faster queries and can include keys from embedded documents and arrays.

#### High Availability

To provide high availability, MongoDB’s replication facility, called replica sets, provide:

* automatic failover.
* data redundancy.

A [replica set](http://docs.mongodb.org/manual/core/replication-introduction/#replication-introduction) is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.

#### Automatic Scaling

MongoDB provides horizontal scalability as part of its core functionality.

* Automatic [sharding](http://docs.mongodb.org/manual/core/sharding-introduction/#sharding-introduction) distributes data across a cluster of machines.
* Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

# Sharding Introduction

Sharding is a method for storing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

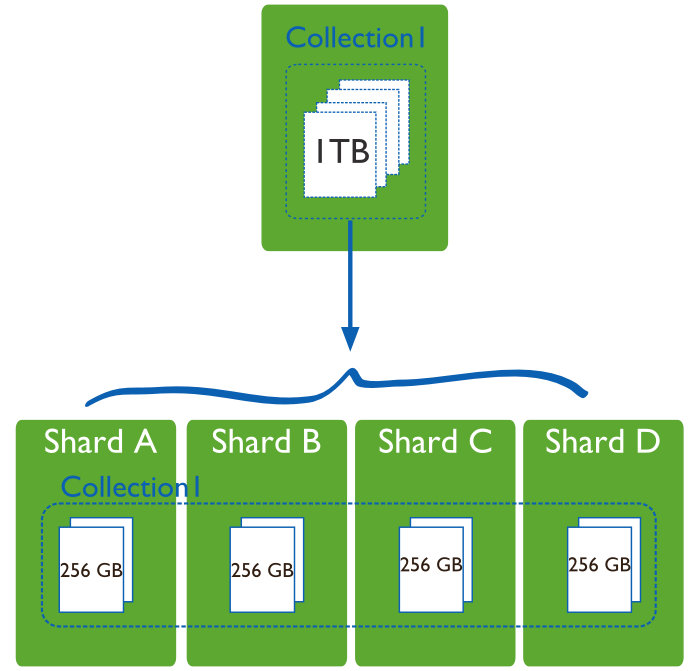
## Purpose of Sharding

Database systems with large data sets and high throughput applications can challenge the capacity of a single server. High query rates can exhaust the CPU capacity of the server. Larger data sets exceed the storage capacity of a single machine. Finally, working set sizes larger than the system’s RAM stress the I/O capacity of disk drives.

To address these issues of scales, database systems have two basic approaches: **vertical scaling** and **sharding**.

**Vertical scaling** adds more CPU and storage resources to increase capacity. Scaling by adding capacity has limitations: high performance systems with large numbers of CPUs and large amount of RAM are disproportionately more expensive than smaller systems. Additionally, cloud-based providers may only allow users to provision smaller instances. As a result there is a practical maximum capability for vertical scaling.

**Sharding**, or horizontal scaling, by contrast, divides the data set and distributes the data over multiple servers, or **shards**. Each shard is an independent database, and collectively, the shards make up a single logical database.



Sharding addresses the challenge of scaling to support high throughput and large data sets:

* Sharding reduces the number of operations each shard handles. Each shard processes fewer operations as the cluster grows. As a result, a cluster can increase capacity and throughput horizontally.

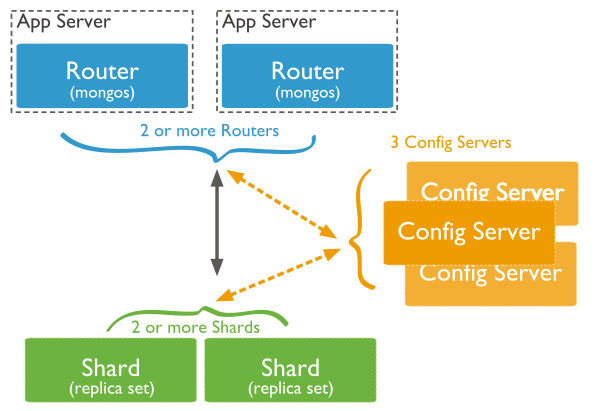
For example, to insert data, the application only needs to access the shard responsible for that record.

* Sharding reduces the amount of data that each server needs to store. Each shard stores less data as the cluster grows.

For example, if a database has a 1 terabyte data set, and there are 4 shards, then each shard might hold only 256GB of data. If there are 40 shards, then each shard might hold only 25GB of data.

## Sharding in MongoDB

MongoDB supports sharding through the configuration of a [sharded clusters](http://docs.mongodb.org/manual/reference/glossary/#term-sharded-cluster).



# Install MongoDB on Windows

<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/>

# Getting Started with MongoDB

<http://docs.mongodb.org/manual/tutorial/getting-started/>

# Generate Test Data

<http://docs.mongodb.org/manual/tutorial/generate-test-data/>

Top of Form

Bottom of Form

[Download the Whitepaper](http://www.mongodb.com/lp/white-paper/nosql-considerations)

## NoSQL vs. SQL Summary

|  | **SQL Databases** | **NoSQL Databases** |
| --- | --- | --- |
| Types | One type (SQL database) with minor variations | Many different types including key-value stores, [document databases](http://www.mongodb.com/document-database), wide-column stores, and graph databases |
| Development History | Developed in 1970s to deal with first wave of data storage applications | Developed in 2000s to deal with limitations of SQL databases, particularly concerning scale, replication and unstructured data storage |
| Examples | MySQL, Postgres, Oracle Database | MongoDB, Cassandra, HBase, Neo4j |
| Data Storage Model | Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Separate data types are stored in separate tables, and then joined together when more complex queries are executed. For example, "offices" might be stored in one table, and "employees" in another. When a user wants to find the work address of an employee, the database engine joins the "employee" and "office" tables together to get all the information necessary. | Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ("key" and "value"), with more complex information sometimes stored within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single "document" in JSON, XML, or another format, which can nest values hierarchically. |
| Schemas | Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline. | Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically. |
| Scaling | Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required. | Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary |
| Development Model | Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database) | Open-source |
| Supports Transactions | Yes, updates can be configured to complete entirely or not at all | In certain circumstances and at certain levels (e.g., document level vs. database level) |
| Data Manipulation | Specific language using Select, Insert, and Update statements, e.g. SELECT fields FROM table WHERE… | Through object-oriented APIs |
| Consistency | Can be configured for strong consistency | Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra) |