**What are the advantages in MongoDB over SQL?**

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB there is no concept of relationship.  
  
Advantages of MongoDB over RDBMS

* Schema less : MongoDB is document database in which one collection holds different different documents. Number of fields, content and size of the document can be differ from one document to another.
* Structure of a single object is clear
* No complex joins
* Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL
* Tuning
* Ease of scale-out: MongoDB is easy to scale
* Conversion / mapping of application objects to database objects not needed
* Uses internal memory for storing the (windowed) working set, enabling faster access of data

Why should use MongoDB

* Document Oriented Storage : Data is stored in the form of JSON style documents
* Index on any attribute
* Replication & High Availability
* Auto-Sharding
* Rich Queries
* Fast In-Place Updates
* Professional Support By MongoDB

Where should use MongoDB?

* Big Data
* Content Management and Delivery
* Mobile and Social Infrastructure
* User Data Management
* Data Hub

Consider you are having table for college students in mysql having fields student name, roll number, class, marks and vehicle number. But every student may not have vehicle in such situations memory will get to allocated to those students vehicle fields as well as you are using mysql i.e. you are defining those fields previously only.  
But in case of mongoDB as you are inserting records as per your requirements in respective collection there is no need to define fields at the time of table creation which indirectly saves your memory in such situations.

MongoDB is based on concept of 'NoSQL'. Which means, it does not use conventional SQL methods & ways to store the data & keep the relation. Learn more here: [NoSQL Databases](http://www.mongodb.com/nosql-explained)

**Benefits of NoSQL**

Currently, persistent data in applications is stored in some external storage like a local network or cloud file system for unstructured data or, in the case of structured data, a database. The role of the database is usually handled by a [relational database management system](http://en.wikipedia.org/wiki/Relational_database_management_system)(RDBMS).

Wikipedia states, “RDBMSs have become a predominant choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data, and much more since the 1980s”.

But, is there a need for a different way to store and retrieve data?

**Not Only SQL**

RDBMSs operate with a relational model defined by [schema](http://en.wikipedia.org/wiki/Database_schema), where each table is a strictly defined collection of rows and columns and a relationship can then be established between each row in one table and a row in another table. Relational data can be queried and manipulated by using [SQL](http://en.wikipedia.org/wiki/SQL) query language.

But what if it is inconvenient to store data in the form of table(s) or we have other kind of relationships between records and want to quickly access the data? The emerging alternative is [NoSQL](http://en.wikipedia.org/wiki/NoSQL).

Again, let’s use the definition from Wikipedia, “A NoSQL or Not Only SQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.”

It seems NoSQL is characterized more by what **it is not**as there is no strict technical definition of what it is and how to implement it. Still, there are some shared features present in most NoSQL database solutions:

* Non-relational and schema-less data model
* Low latency and high performance
* Highly scalable

Different NoSQL solutions seem to focus on different sets of features, and these solutions have been rapidly increasing over the last few years.

But first let's examine why people stray from RDBMS.

**What is SQL Not Good For?**

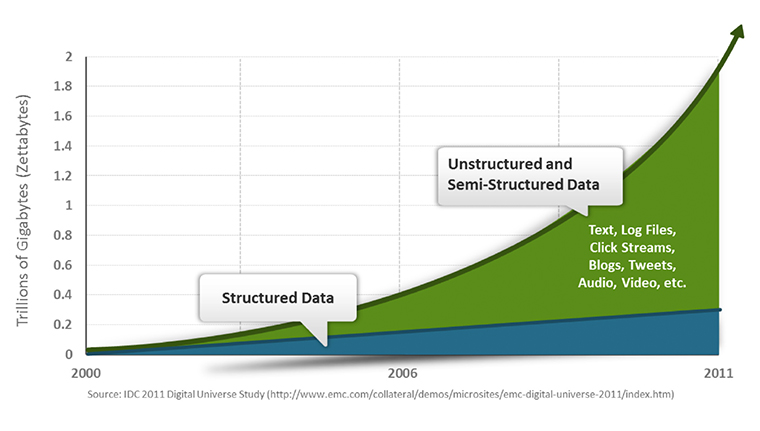
Arguably the biggest problem for developers using relational databases is the [object-relational impedance mismatch](http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch). SQL queries are not well suited for the object oriented data structures that are used in most applications now.

Another closely related issue is storing or retrieving an object with all relevant data. Some application operations require multiple and/or very complex queries. In that case, data mapping and query generation complexity raises too much and becomes difficult to maintain on the application side.

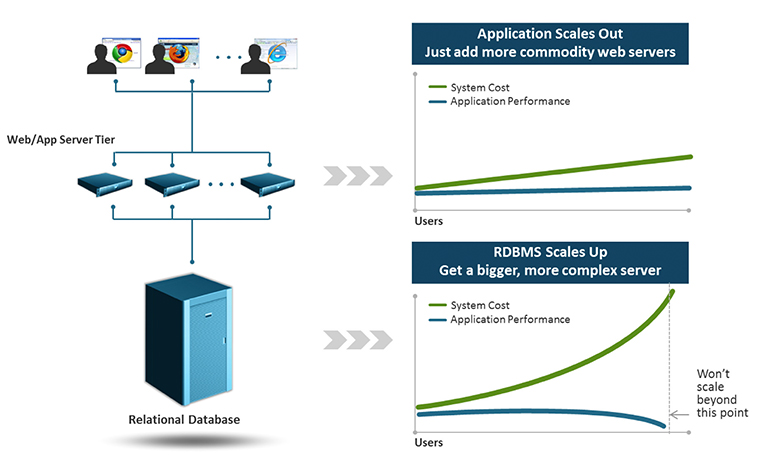
Some of these problems may be tempered by various [Object-relational mapping](http://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) frameworks, but it still requires a lot of development effort to work around most of performance and data access complexity issues.

For various reasons, an alternative, [object databases](http://en.wikipedia.org/wiki/Object_database) (OODBMS), did not gain much popularity in replacing relational databases. However, most object oriented databases may be considered NoSQL solutions too, and may even be reinvigorated on the wave of rising NoSQL popularity.

Another set of problems that relational databases struggle with is related to an exponentially increasing amount of data. The direct consequence is the so-called [big data](http://en.wikipedia.org/wiki/Big_data) problem. This problem arises when standard SQL query operations do not have acceptable performances, especially when transactions are involved.



Trying to cope with such a large amount of data by scaling RDBMS servers leads to configuration and maintainability issues.



Now let’s take a look at what NoSQL has to offer.

**What NoSQL is Good For**

A major difference from relational databases is the lack of explicit data scheme. NoSQL databases infer scheme from stored data, if it requires it at all, depending on which model was used.

The main benefit of using different data models is that they are very good at what they do. At the same time, don’t force them to do something they aren’t designed for. This means that it is of the upmost importance to understand and correctly use the data model when choosing NoSQL solutions.

Generally, data models in NoSQL are grouped into four categories. However, particular NoSQL solutions may incorporate several models at once.

**Key-Value (K-V) Stores**

K-V store is the simplest data model. Technically it is just a distributed persistent[associative array](http://en.wikipedia.org/wiki/Associative_array). The key is a unique identifier for a value, which can be any data application needs stored.

This model is also the fastest way to get data by known key, but without the flexibility of more advanced querying.

It may be used for data sharing between application instances like distributed cache or to store user session data.

**Document Stores**

Document store is a data model for storing semi-structured document object data and metadata. The [JSON](http://en.wikipedia.org/wiki/JSON) format is normally used to represent such objects.

Documents can be queried by their properties in a similar manner to relational databases but aren’t required to adhere to the strict structure of a database table. Additionally, only parts of the object may be requested or updated.

Generally speaking, document stores are used for aggregate objects that have no shared complex data between them and to quickly search or filter by some object properties.

**Column-Oriented Stores**

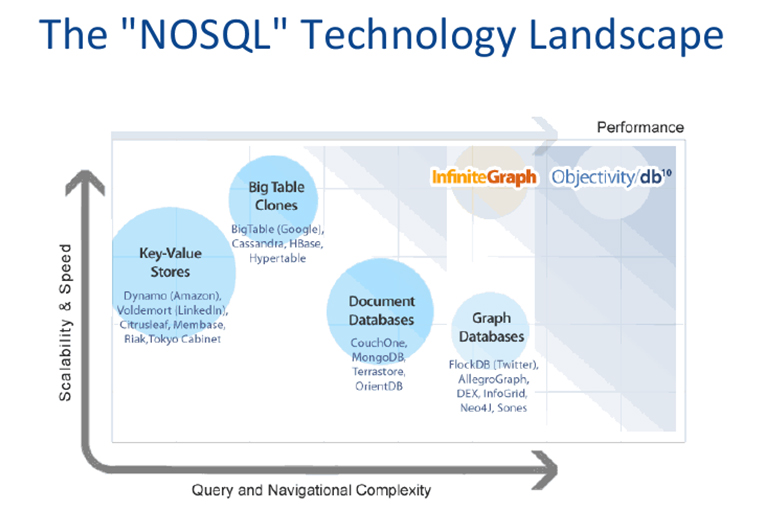
A more advanced K-V store data model is a column family. These are used for organizing data based on individual columns where actual data is used as a key to refer to whole data collections. It is similar to a relational database index, however a column family may be an arbitrary collection of columns. There are more complex aggregation structures like super columns and super column families to allow access to the data by several keys.

This particular approach is used for very large scalable databases to greatly reduce time for searching data. It is rarely used outside of enterprise level applications.

**Graph Databases**

As the name implies, this data model allows objects to link and be linked by several other objects thus constructing a graph structure. Links usually have additional properties to describe the relation between objects.

Graph databases map more directly to object oriented programming models and are faster for highly associative data sets and graph queries. Furthermore they typically support [ACID](http://en.wikipedia.org/wiki/ACID) transaction properties in the same way as most RDBMS.



**Transactions and Consistency in NoSQL**

Many NoSQL solutions compromise consistency (in the sense of the [CAP theorem](http://en.wikipedia.org/wiki/CAP_theorem)) in favor of availability, scalability and partition tolerance. On the other hand, some NoSQL solutions may allow you to specify what level of consistency should be applied for particular operation and some even fully support ACID transactions.

However in the case of key-value or document store data models, transaction consistency is rarely needed as most operations are by definition atomic.

**Quick Preview of Some NoSQL Solutions**

Here is a quick preview of some NoSQL solutions we are planning to use or already trying out. We are also particularly interested in integrating them with Azure Cloud.

**Redis**

Redis, *RE*mote *DI*ctionary *S*erver, it is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets.

Redis allows a user to set an expiration time for key-value pairs and requires all stored data to fit into a server RAM. Clearly, it is designed to be used as a distributed caching and session service.

Data storage in RAM allows very fast read/write operations. Furthermore, data is persisted to a disk and in the case of a server restart can be restored back to RAM for quick access.

The approximate memory usage provided by Redis developers are:

* An empty instance uses – 1MB of memory
* 1 Million Keys – String Value pairs use 100MB of memory
* 1 Million Keys – Hash value, representing an object with 5 fields, use 200 MB of memory

Various useful atomic operations are supported like increment and decrement for integer values.

**MongoDB**

MongoDB is document storage designed for high performance, high availability, and with automatic scaling.

Documents are saved in a BSON format (binary JSON) and field values aside from the usual JSON types can include other documents, arrays and arrays of documents. Every field can be indexed and queried.

MongoDB has a write lock support which blocks all other operations, including reads.

Also, MangoDB supports dynamic consistency where each write operation can specify the guaranteed level of success for that operation. When inserts, updates and deletes have a weak write concern, write operations return quickly. In some failed cases, write operations issued with weak write concerns may not continue. With stronger write concerns, clients wait after sending a write operation for MongoDB to confirm the write operations.

Additional notable features include:

* Geospatial indexing allowing location based queries
* GridFS for very large file support

MangoDB is able to be used as primary storage for CMS content.

**CouchDB**™

Apache CouchDB is a document storage mainly targeted for mobile devices with offline mode support.

It uses JSON for document storage and REST for API. Field values are restricted to standard JSON types.

CouchDB provides ACID transaction semantics meaning that it can handle a high volume of concurrent readers and writers without conflict. CouchDB also guarantees eventual consistency to be able to provide both availability and partition tolerance.

Aggregation in CouchDB is done by using a specialized view model similar to [a map-reduce system](http://en.wikipedia.org/wiki/MapReduce), and is continuously updated and processed in parallel.

CouchDB is a perfect candidate for usage on mobile devices and client side focused web browser applications.

**Final Words**

There is a need to emphasize the fact that NoSQL solutions will not offer some “miracle cure” for all application data handling requirements. These are tools designed for specific purposes with their own advantages and disadvantages.

**What exactly is NoSQL? Where is it used actually?**

**What is it?**  
NoSQL refers to a general class of storage engines that store data in a non-relational format.  This is in contrast to traditional RDBMS in which data is stored in tables that have data that relate to each other.  NoSQL stands for "Not Only SQL" and isn't meant as a rejection of traditional databases.  
  
**What kinds of NoSQL databases are out there?**  
There's different kinds of NoSQL databases for different jobs.  I tend to think of there being about 4 kinds of them available today:

1. *Key-Value Stores*: Are very simple in that you simply define a key for a binary object.  Its very common for programmers to simple store large serialized objects in these kinds of DBs.  Examples are [Cassandra (database)](https://www.quora.com/topic/Cassandra-database), and Oracle NoSQL.
2. *Document Store*: Stores "documents" also based on a key-value system although more structured.  The most common implementation is based on the JSON (JavaScript Object Notation) standard, which I tend to think of as a similar structure to XML.  Examples are [MongoDB](https://www.quora.com/topic/MongoDB) and [CouchDB](https://www.quora.com/topic/CouchDB" \t "_blank).
3. *Graph DB:* Stores data as "graphs" which allow you to define complex relationships between objects.  Very common for something like storing relationships of people in a social network.  Examples are [Neo4j](https://www.quora.com/topic/Neo4j).
4. *Column Oriented:*  Data is stored in columns rather than rows (this is a tricky concept to get at first).  Allows for great compression and for building tables that are very very large (hundreds of thousands of columns, billions/trillions of rows).  Examples are[HBase](https://www.quora.com/topic/HBase).

**How do you use them?**  
Generally speaking, you can't just connect to a NoSQL database and start running queries like you can with RDBMS.  NoSQL database tend to be the pure storage engine and rely on the application to do any kind of data manipulation.  For example, most NoSQL DB's don't provide something like a simple SUM() function.  In most cases, you use a programming API to make calls to the NoSQL DB and retrieve data (value) based on a key.  In general, data in a NoSQL database is accessed 1 "row"/object at a time.  
  
**So what do you use them for?**  
A lot of things.  In general, NoSQL databases excel when you need something that can both read and write large amounts of data quickly.  And since they scale horizontally, just adding more servers tends to improve performance with little effort.  Facebook uses it for you Inbox.  Other examples might be a user's game online profile or storing large amounts of legal documents.  An RDBMS is still the best option for handling large numbers of atomic level transactions (IE, we likely won't see things like banking systems or supply chain management systems run on a NoSQL database).  This is also because they are not ACID compliant (basically two people looking at the same key might see different values, depending on when the data was accessed).