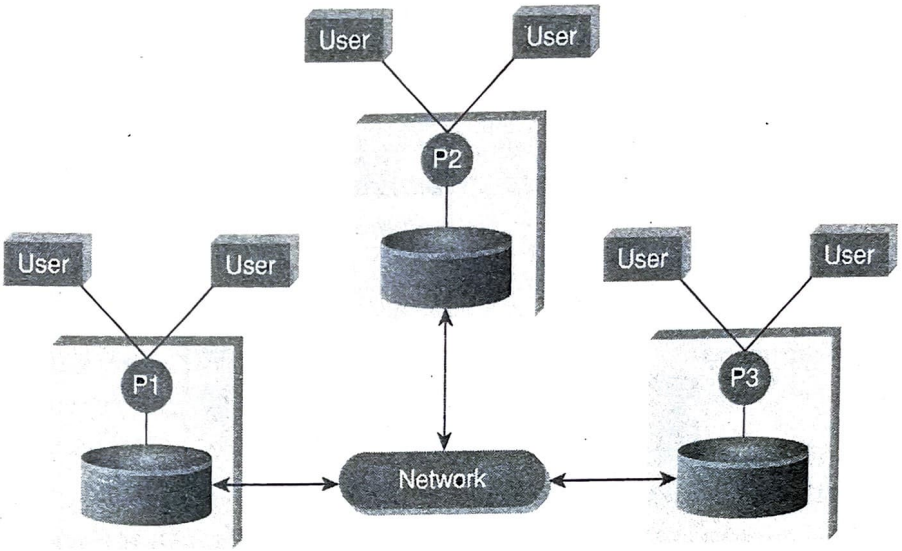


Figure 3.12 Distributed system.

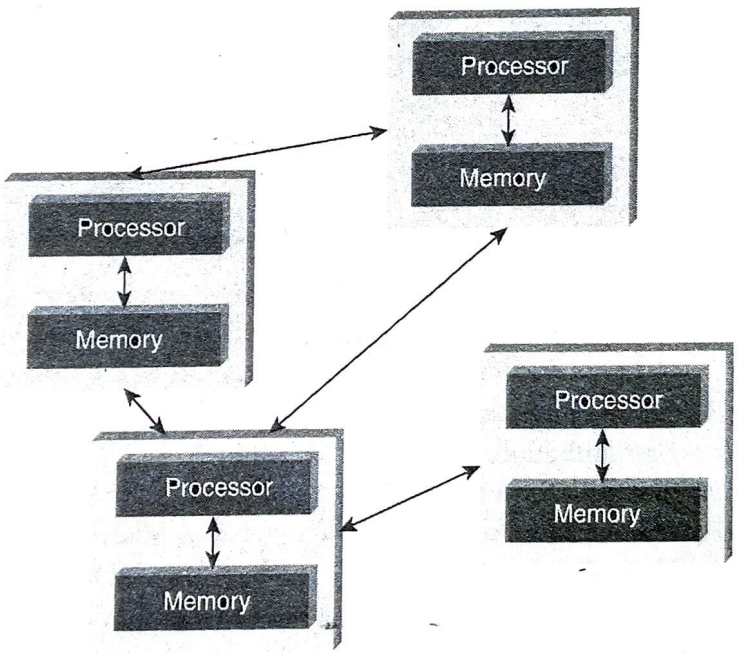


Figure 3.13 Distributed system.

3.12.6.1 Advantages of a "Shared Nothing Architecture"

1. **Fault Isolation:** A "Shared Nothing Architecture" provides the benefit of isolating fault. A fault in a single node is contained and confined to that node exclusively and exposed only through messages (or lack of it).
2. **Scalability:** Assume that the disk is a shared resource. It implies that the controller and the disk bandwidth are also shared. Synchronization will have to be implemented to maintain a consistent shared state. This would mean that different nodes will have to take turns to access the critical data. This

imposes a limit on how many nodes can be added to the distributed shared disk system, thus compromising on scalability.

3.12.7 CAP Theorem Explained

The CAP theorem is also called the *Brewer's Theorem*. It states that in a distributed computing environment (a collection of interconnected nodes that share data), it is impossible to provide the following guarantees. Refer Figure 3.14. At best you can have two of the following three – one must be sacrificed.

1. Consistency
2. Availability
3. Partition tolerance

3.12.7.1 CAP Theorem

Let us spend some time understanding the earlier mentioned terms.

1. Consistency implies that every read fetches the last write.
2. Availability implies that reads and writes always succeed. In other words, each non-failing node will return a response in a reasonable amount of time.
3. Partition tolerance implies that the system will continue to function when network partition occurs.

Let us try to understand this using a real-life situation.

You work for a training institute, "XYZ." The institute has 50 instructors including you. All of you report to a training coordinator. At the end of the month, all the instructors together with the training coordinator peruse through the training requests received from the various corporate houses and prepare a training schedule for each instructor. These training schedules (one for each instructor) are shared with "Amey," the office administrator. Each morning, you either call the office helpdesk (essentially Amey's desk) or check in-person with Amey for your schedule for the day. In case a training request has been cancelled or updated (updates can be in the form of change in course, change in duration, change of the training timings, etc.), Amey is informed of the updates and the schedules are subsequently updated by him.

Things were good until now. Few corporate houses were your clients and the schedules of each instructor could be smoothly managed without any major hiccups. But your training institute has been implementing promotion campaigns to expand the business. As a result of advertising in the media and word of mouth publicity by your existing clients, you suddenly see an upsurge in training requests from existing and new clients. In consequence of that, more instructors have been recruited. Few trainers/consultants have also been roped in from other training institutes to help tackle the load.

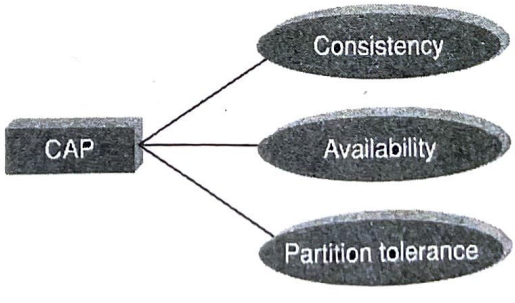


Figure 3.14 Brewer's CAP.

Now when you go to Amey to check your schedule or call in at the helpdesk, you are prepared for a wait in the queue. Looking at the current state of affairs, the training coordinator decides to recruit an additional office administrator "Joey." The helpdesk number will remain the same and will be shared by both the office administrators.

This arrangement works well for a couple of days. Then one day…

You: Hey Amey!

Amey: Hi! How can I help?

You: I think I am scheduled to anchor a training at 3:00 pm today. Can I please have the details?

Amey: Sure! Just a minute.

Amey browses through the file where he maintains the schedules. He does not see a training scheduled against your name at 3:00 pm today and responds back, "You do not have any training to conduct at 3:00 pm."

You: How is that possible? The training coordinator called up yesterday evening to inform of the same and said he has updated the office administrators of the same.

Amey: Oh! Did he say which office administrator? It could have been Joey. Please check with Joey.

Amey: Hey Joey! Please check the schedule for Paul here… Do you see something scheduled at 3:00 pm today?

Joey: Sure enough! He is anchoring the training for client "Z" today at 3:00 pm.

A clear case of inconsistent system!!! The updates in the schedule were shared by the training coordinator with Joey and you were checking for your schedule with Amey.

You share this incident with the training coordinator and that gets him thinking. The issue has to be addressed immediately otherwise it will be difficult to avoid a chaotic situation. He comes up with a plan and shares it with both the office administrators the following day.

Training Coordinator: Folks, each time that either an instructor or me calls any one of you to update a schedule, make sure that both of you update it in your respective files. This way the instructor will always get the most recent and consistent information irrespective of whom amongst the two of you he/she speaks to.

Joey: But that could mean a delay in answering either a phone call or sharing the schedule with the instructor waiting in queue.

Training Coordinator: Yes, I understand. But there is no way that we can give incorrect information.

Amey: There is this other problem as well. Suppose one of us is on leave on a particular day. That would mean that we cannot take any update related calls as we will not be able to simultaneously update both the files (my file and Joey's).

Training Coordinator: Well, good point! *That's the availability problem!!!* But I have thought about that as well. Here is the plan:

1. If one of you receives the update call (any updates to any schedule), ensure that you inform the other person if he is available.
2. In case the other person is not available, ensure that you inform him of all the updates to all schedules via email. It is a must!!!
3. When the other person resumes duty, the first thing he will do is update his file with all the updates to all schedules that he has received via email.

Wow!!! That is sure a Consistent and Available system!!!

Looks like everything is in control. Wait a minute! There is a tiff that has taken place between the office administrators. The two are pretty much available but are not talking to each other which, in other words, means that the updates are not flowing from one to the other. *We have to be partition tolerant!!!* As a training coordinator, you instruct them saying that none of you are taking any calls requesting for schedules or updates to schedules till you patch up. This implies that the system is partition tolerant but not available at that time.

In summary, one can at most decide to go with two of the three.

1. **Consistent:** The instructors or the training coordinator, once they have updated information with you, will always get the most updated information when they call subsequently.
2. **Availability:** The instructors or the training coordinators will always get the schedule if any or both of the office administrators have reported to work.
3. **Partition Tolerance:** Work will go on as usual even if there is communication loss between the office administrators owing to a spat or a tiff!

When to choose consistency over availability and vice-versa…

1. Choose availability over consistency when your business requirements allow some flexibility around when the data in the system synchronizes.
2. Choose consistency over availability when your business requirements demand atomic reads and writes.

Examples of databases that follow one of the possible three combinations:

1. Availability and Partition Tolerance (AP)
2. Consistency and Partition Tolerance (CP)
3. Consistency and Availability (CA)

C —
A —
P — Partition tolerance

Refer Figure 3.15 to get a glimpse of databases that adhere to two of the three characteristics of CAP theorem.

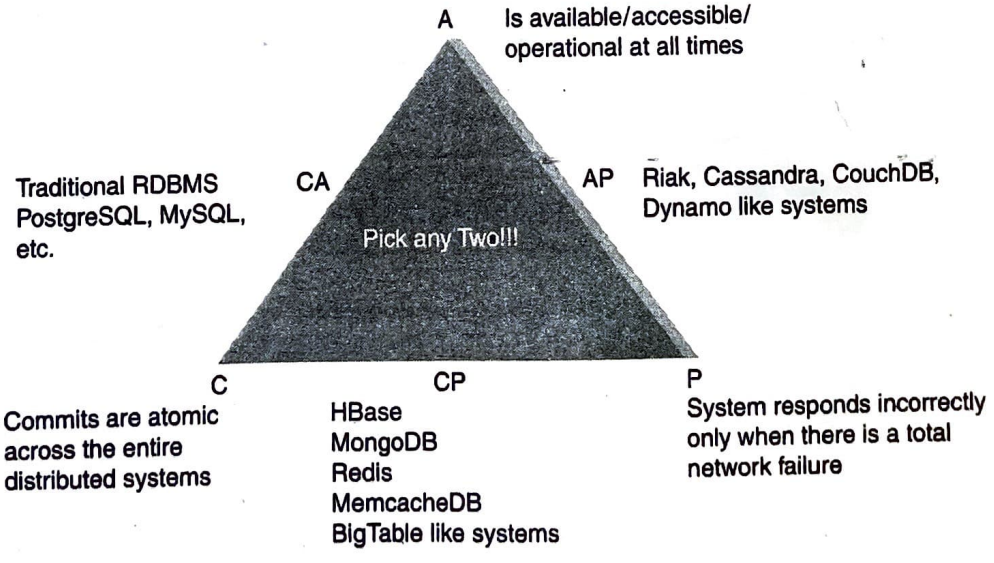


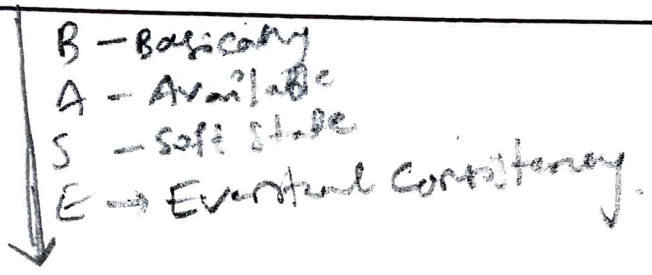
Figure 3.15 Databases and CAP.

ICALLY AVAILABLE SOFT STATE EVENTUAL CONSISTENCY ;E)

stions to start with:

used?
:d computing.

:ed?
ıigh availability.



:hieved?
ven data item. If no new updates are made to this given data item for a stipulated period of ıally all accesses to this data item will return the updated value. In other words, if no new made to a given data item for a stipulated period of time, all updates that were made in the t applied to this given data item and the several replicas of it will percolate to this data item .ys as current/recent as is possible.

›lica convergence?
ıt has achieved eventual consistency is said to have converged or achieved *replica convergence*.

***·olution*: How is the conflict resolved?**
pair: If the read leads to discrepancy or inconsistency, a correction is initiated. It slows down . operation.
epair: If the write leads to discrepancy or inconsistency, a correction is initiated. This will ıe write operation to slow down.
ronous repair: Here, the correction is not part of a read or write operation.

V TOP ANALYTICS TOOLS

arth of analytical tools in the market. Please find below our list of few top analytics tools. rovided the links after each tool for you to explore more…

l
ıpport.office.microsoft.com/en-in/article/Whats-new-in-Excel-2013-1cbc42cd-bfaf-43d7-88ef1392fd?CorrelationId=1a2171cc-191f-47de-8a55-08a5f2e9c739&ui=en-US&rs=en-IN

ww.sas.com/en_us/home.html

;S Modeler
ww-01.ibm.com/software/analytics/spss/products/modeler/

5. Salford systems (World Programming Systems)
   http://www.salford-systems.com/
6. WPS
   http://www.teamwpc.co.uk/products/wps

3.14.1 Open Source Analytics Tools

Let us look at a couple of open source analytics tools. We have also provided the links after each tool for you to explore more…

1. R analytics
   http://www.revolutionanalytics.com/
2. Weka
   http://www.cs.waikato.ac.nz/ml/weka/

REMIND ME

- Quite a few data analytics and visualization tools are available in the market today from leading vendors such as IBM, Tableau, SAS, R Analytics, Statistica, World Programming Systems (WPS), etc. to help process and analyze your big data.
- Big data analytics is a about a tight handshake between three communities: IT, business users, and data scientists.
- *Data science* is the science of extracting knowledge from data.
- The CAP theorem is also called the Brewer's Theorem. It states that in a distributed computing environment (a collection of interconnected nodes that share data), it is impossible to provide the following guarantees. At best you can have two of the following three – one must be sacrificed.
  - Consistency
  - Availability
  - Partition tolerance

CONNECT ME (INTERNET RESOURCES)

- http://en.wikipedia.org/wiki/Data_science
- http://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/
- http://www.oralytics.com/2012/06/data-science-is-multidisciplinary.html
- http://spotfire.tibco.com/blog/?p=4240
- http://reports.informationweek.com/abstract/106/1255/Financial/tech-center-taking-advantage-of-in-memory-analytics.html
- http://www.informationweek.com/software/information-management/oracle-analytics-package-

The big data technology landscape can be majorly studied under two important technologies:

1. NoSQL
2. Hadoop

4.1 NoSQL (NOT ONLY SQL)

The term NoSQL was first coined by Carlo Strozzi in 1998 to name his lightweight, open-source, non-relational database that did not expose the standard SQL interface. The term was reintroduced by Eric Evans in early 2009.

4.1.1 Where is it Used?

NoSQL databases are widely used in big data and other real-time web applications. Refer Figure 4.1. NoSQL databases is used to stock log data which can then be pulled for analysis. Likewise it is used to store social media data and all such data which cannot be stored and analyzed comfortably in RDBMS.

4.1.2 What is it?

NoSQL stands for Not Only SQL. These are non-relational, open source, distributed databases. They are hugely popular today owing to their ability to scale out or scale horizontally and the adeptness at dealing with a rich variety of data: structured, semi-structured and unstructured data. Refer Figure 4.2 for additional features of NoSQL.

1. **NoSQL databases are non-relational:** They do not adhere to relational data model. In fact, they are either key-value pairs or document-oriented or column-oriented or graph-based databases.
2. **Distributed:** They are distributed meaning the data is distributed across several nodes in a cluster constituted of low-cost commodity hardware.

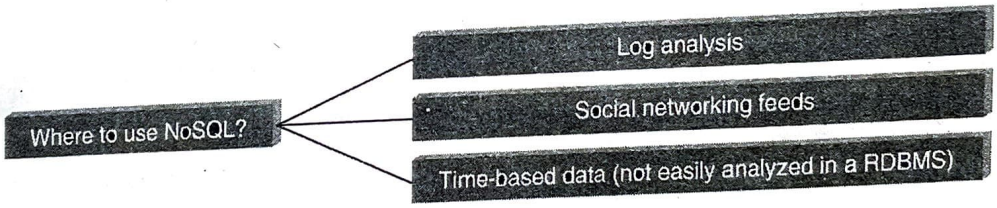


Figure 4.1 Where to use NoSQL?

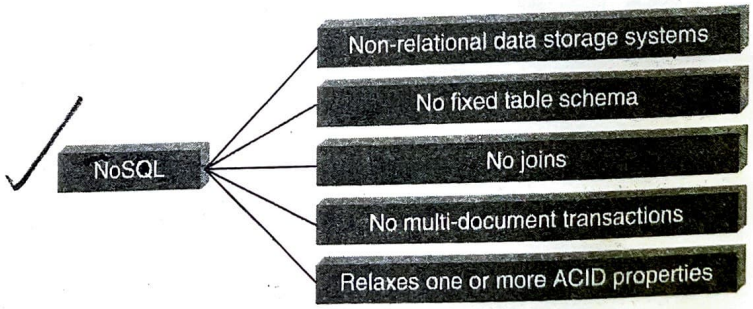


Figure 4.2 What is NoSQL?

3. **No support for ACID properties (Atomicity, Consistency, Isolation, and Durability):** They do not offer support for ACID properties of transactions. On the contrary, they have adherence to Brewer's CAP (Consistency, Availability, and Partition tolerance) theorem and are often seen compromising on consistency in favor of availability and partition tolerance.
4. **No fixed table schema:** NoSQL databases are becoming increasing popular owing to their support for flexibility to the schema. They do not mandate for the data to strictly adhere to any schema structure at the time of storage.

4.1.3 Types of NoSQL Databases

We have already stated that NoSQL databases are non-relational. They can be broadly classified into the following:

1. Key–value or the big hash table.
2. Schema-less.

Refer Figure 4.3. Let us take a closer look at key–value and few other types of schema-less databases:

1. **Key–value:** It maintains a big hash table of keys and values. For example, Dynamo, Redis, Riak, etc.

Sample Key–Value Pair in Key–Value Database

Key	Value
First Name	Simmonds
Last Name	David

2. **Document:** It maintains data in collections constituted of documents. For example, MongoDB, Apache CouchDB, Couchbase, MarkLogic, etc.

Sample Document in Document Database

```
{
"Book Name":    "Fundamentals of Business Analytics",
"Publisher":    "Wiley India",
"Year of Publication":    "2011"
}
```

3. **Column:** Each storage block has data from only one column. For example: Cassandra, HBase, etc.

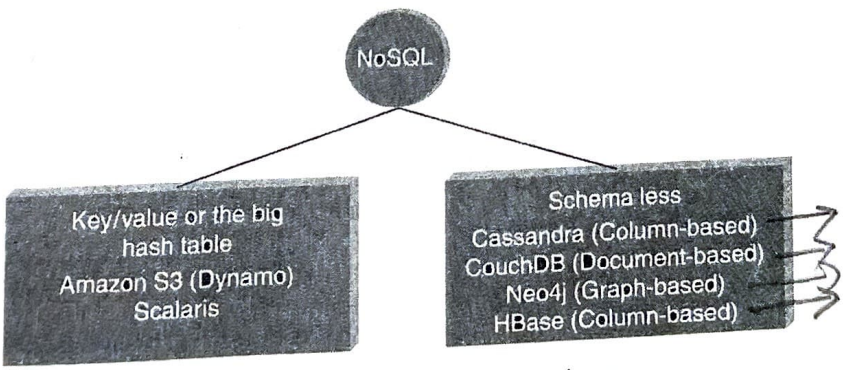
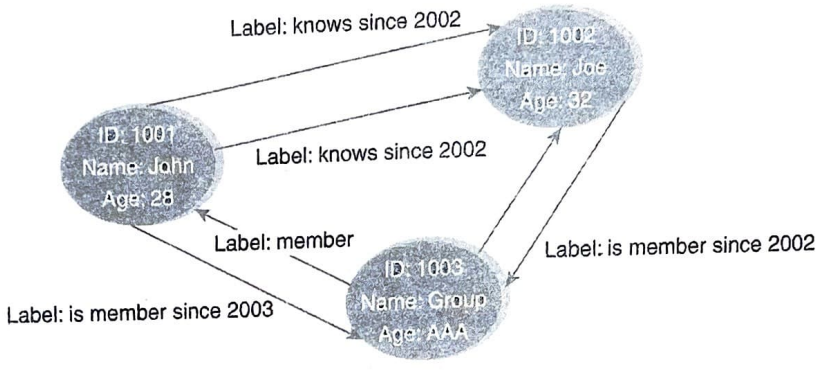


Figure 4.3 Types of NoSQL databases.

4. **Graph:** They are also called network database. A graph stores data in nodes. For example, Neo4j, HyperGraphDB, etc.

Sample Graph in Graph Database



Refer Table 4.1 for popular schema-less databases.

4.1.4 Why NoSQL?

1. It has scale out architecture instead of the monolithic architecture of relational databases.
2. It can house large volumes of structured, semi-structured, and unstructured data.
3. **Dynamic schema:** NoSQL database allows insertion of data without a pre-defined schema. In other words, it facilitates application changes in real time, which thus supports faster development, easy code integration, and requires less database administration.
4. **Auto-sharding:** It automatically spreads data across an arbitrary number of servers. The application in question is more often not even aware of the composition of the server pool. It balances the load of data and query on the available servers; and if and when a server goes down, it is quickly replaced without any major activity disruptions.
5. **Replication:** It offers good support for replication which in turn guarantees high availability, fault tolerance, and disaster recovery.

4.1.5 Advantages of NoSQL

Let us enumerate the advantages of NoSQL. Refer Figure 4.4.

1. **Can easily scale up and down:** NoSQL database supports scaling rapidly and elastically and even allows to scale to the cloud.

Table 4.1 Popular schema-less databases

Key–Value Data Store	Column-Oriented Data Store	Document Data Store	Graph Data Store
• Riak	• Cassandra	• MongoDB	• InfiniteGraph
• Redis	• HBase	• CouchDB	• Neo4j
• Membase	• HyperTable	• RavenDB	• AllegroGraph

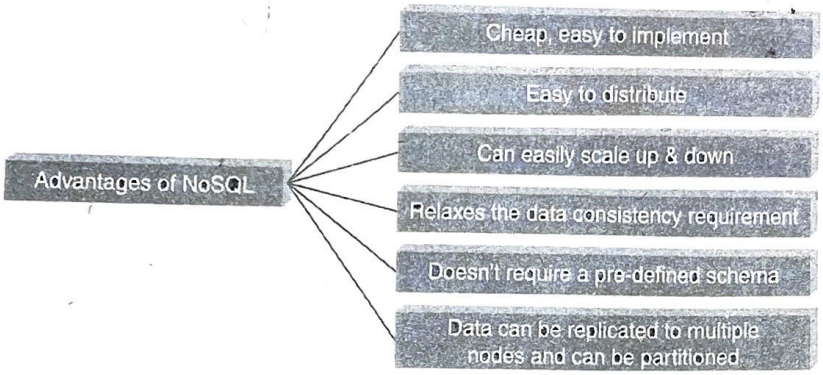


Figure 4.4 Advantages of NoSQL.

(a) ***Cluster scale:*** It allows distribution of database across 100+ nodes often in multiple data centers.
(b) ***Performance scale:*** It sustains over 100,000+ database reads and writes per second.
(c) ***Data scale:*** It supports housing of 1 billion+ documents in the database.

2. **Doesn't require a pre-defined schema:** NoSQL does not require any adherence to pre-defined schema. It is pretty flexible. For example, if we look at MongoDB, the documents (equivalent of records in RDBMS) in a collection (equivalent of table in RDBMS) can have different sets of key-value pairs.

   {_id: 101,"BookName": "Fundamentals of business analytics", "AuthorName": "Seema Acharya", "Publisher": "Wiley India"}
   {_id:102, "BookName":"Big Data and Analytics"}

3. **Cheap, easy to implement:** Deploying NoSQL properly allows for all of the benefits of scale, high availability, fault tolerance, etc. while also lowering operational costs.
4. **Relaxes the data consistency requirement:** NoSQL databases have adherence to CAP theorem (Consistency, Availability, and Partition Tolerance). Most of the NoSQL databases compromise on consistency in favor of availability and partition tolerance. However, they do go for eventual consistency.
5. **Data can be replicated to multiple nodes and can be partitioned:** There are two terms that we will discuss here:

   (a) ***Sharding:*** Sharding is when different pieces of data are distributed across multiple servers. NoSQL databases support auto-sharding meaning they can natively and automatically spread data across an arbitrary number of servers, without requiring the application to even be aware of the composition of the server pool. Servers can be added or removed from the data layer without application downtime. This would mean that data and query load are automatically balanced across servers, and when a server goes down, it can be quickly and transparently replaced with no application disruption.
   (b) ***Replication:*** Replication is when multiple copies of data are stored across the cluster and even across data centers. This promises high availability and fault tolerance.

4.1.6 What We Miss With NoSQL?

With NoSQL around, we have been able to counter the problem of scale (NoSQL scales out). There is also the flexibility with respect to schema design. However there are few features of conventional RDBMS that are greatly missed. Refer Figure 4.5.

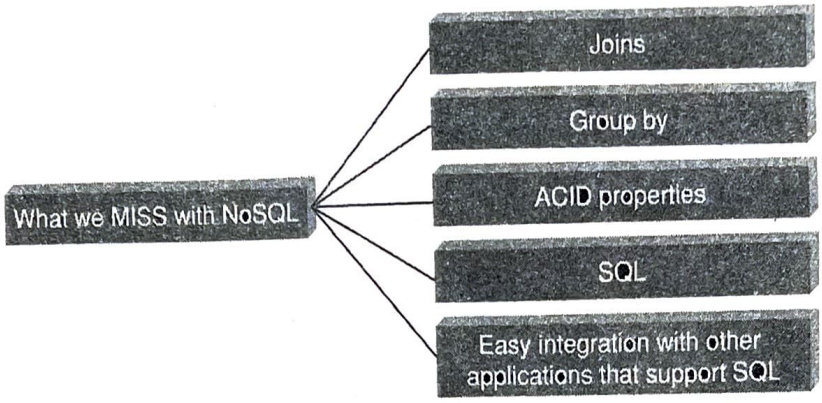


Figure 4.5 What we miss with NoSQL?

NoSQL does not support joins. However, it compensates for it by allowing embedded documents as in MongoDB. It does not have provision for ACID properties of transactions. However, it obeys the Eric Brewer's CAP theorem. NoSQL does not have a standard SQL interface but NoSQL databases such as MongoDB and Cassandra have their own rich query language [MongoDB query language and Cassandra query language (CQL)] to compensate for the lack of it. One thing which is dearly missed is the easy integration with other applications that support SQL.

4.1.7 Use of NoSQL in Industry

NoSQL is being put to use in varied industries. They are used to support analysis for applications such as web user data analysis, log analysis, sensor feed analysis, making recommendations for upsell and cross-sell, etc. Refer Figure 4.6.

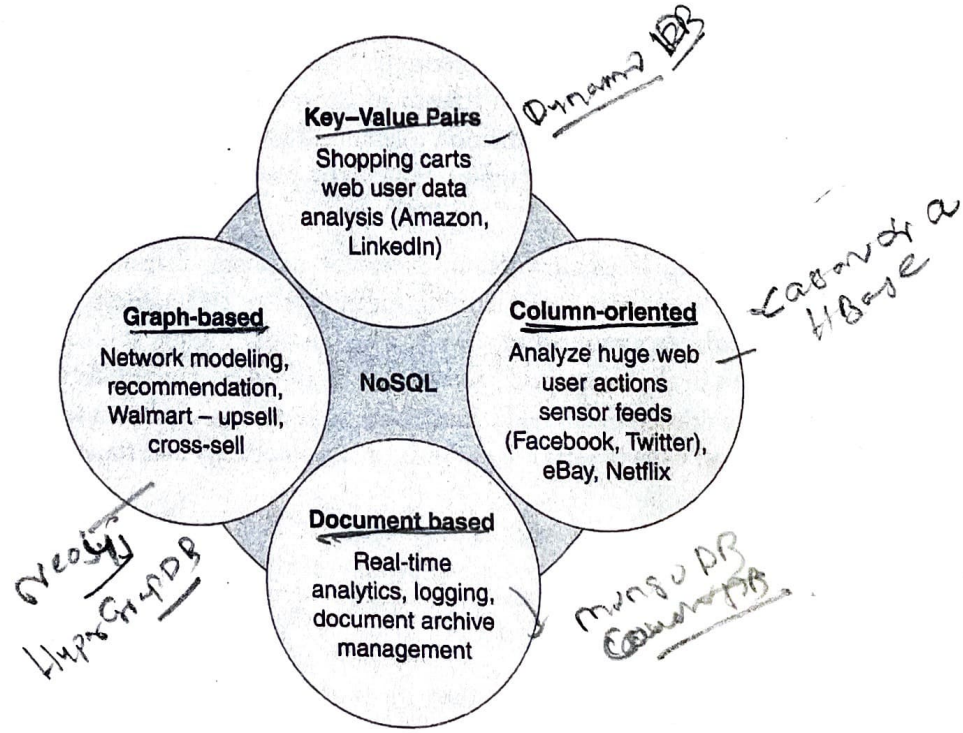


Figure 4.6 Use of NoSQL in industry.

4.1.8 NoSQL Vendors

Refer Table 4.2 for few popular NoSQL vendors.

4.1.9 SQL versus NoSQL

Refer Table 4.3 for few salient differences between SQL and NoSQL.

Table 4.2 Few popular NoSQL vendors

Company	Product	Most Widely Used by
Amazon	DynamoDB	LinkedIn, Mozilla
Facebook	Cassandra	Netflix, Twitter, eBay
Google	BigTable	Adobe Photoshop

Table 4.3 SQL versus NoSQL

SQL	NoSQL
Relational database	Non-relational, distributed database
Relational model	Model-less approach
Pre-defined schema	Dynamic schema for unstructured data
Table based databases	Document-based or graph-based or wide column store or key-value pairs databases
Vertically scalable (by increasing system resources)	Horizontally scalable (by creating a cluster of commodity machines)
Uses SQL	Uses UnQL (Unstructured Query Language)
Not preferred for large datasets	Largely preferred for large datasets
Not a best fit for hierarchical data	Best fit for hierarchical storage as it follows the key-value pair of storing data similar to JSON (Java Script Object Notation)
Emphasis on ACID properties	Follows Brewer's CAP theorem
Excellent support from vendors	Relies heavily on community support
Supports complex querying and data keeping needs	Does not have good support for complex querying
Can be configured for strong consistency	Few support strong consistency (e.g., MongoDB), few others can be configured for eventual consistency (e.g., Cassandra)
Examples: Oracle, DB2, MySQL, MS SQL, PostgreSQL, etc.	MongoDB, HBase, Cassandra, Redis, Neo4j, CouchDB, Couchbase, Riak, etc.

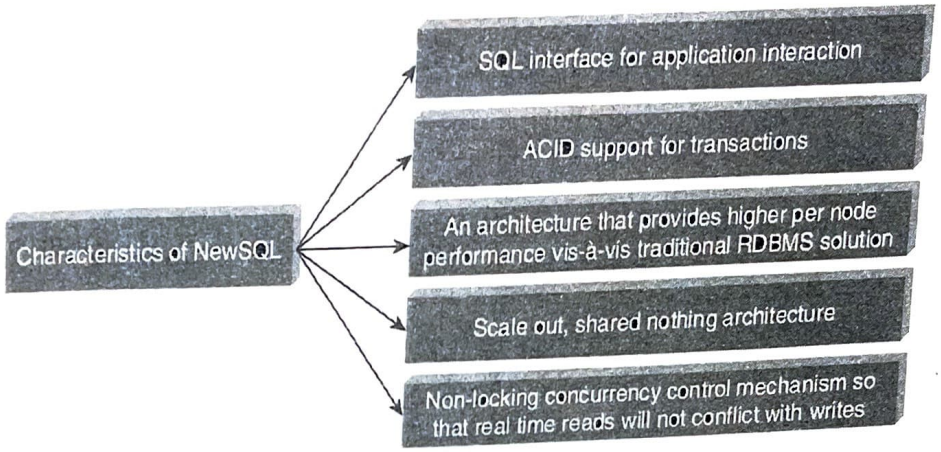


Figure 4.7 Characteristics of NewSQL.

4.1.10 NewSQL

There is yet another new term doing the rounds – "NewSQL". So, what is NewSQL and how is it different from SQL and NoSQL?

What is that we love about NoSQL and is not there with our traditional RDBMS and what is that we love about SQL that NoSQL does not have support for? You guessed it right!!! We need a database that has the same scalable performance of NoSQL systems for On Line Transaction Processing (OLTP) while still maintaining the ACID guarantees of a traditional database. This new modern RDBMS is called NewSQL. It supports relational data model and uses SQL as their primary interface.

4.1.10.1 *Characteristics of NewSQL*

Refer Figure 4.7 to learn about the characteristics of NewSQL. NewSQL is based on the shared nothing architecture with a SQL interface for application interaction.

4.1.11 Comparison of SQL, NoSQL, and NewSQL

Refer Table 4.4 for a comparative study of SQL, NoSQL and NewSQL.

Table 4.4 Comparative study of SQL, NoSQL, and NewSQL

	SQL	NoSQL	NewSQL
Adherence to ACID properties	Yes	No	Yes
OLTP/OLAP	Yes	No	Yes
Schema rigidity	Yes	No	Maybe
Adherence to data model	Adherence to relational model		
Data Format Flexibility	No	Yes	Maybe
Scalability	Scale up Vertical Scaling	Scale out Horizontal Scaling	Scale out
Distributed Computing	Yes	Yes	Yes
Community Support	Huge	Growing	Slowly growing

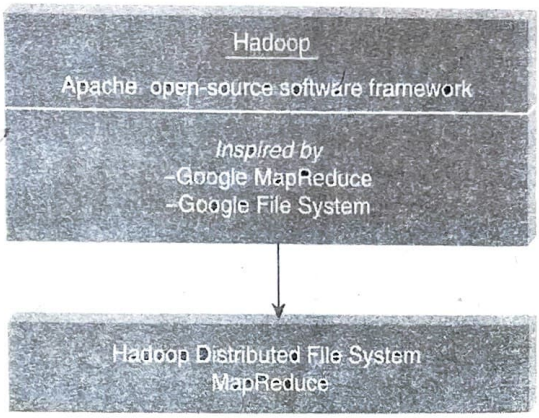


Figure 4.8 Hadoop.

4.2 HADOOP

Hadoop is an open-source project of the Apache foundation. It is a framework written in Java, originally developed by Doug Cutting in 2005 who named it after his son's toy elephant. He was working with Yahoo then. It was created to support distribution for "Nutch", the text search engine. Hadoop uses Google's MapReduce and Google File System technologies as its foundation. Hadoop is now a core part of the computing infrastructure for companies such as Yahoo, Facebook, LinkedIn and Twitter, etc. Refer Figure 4.8.

4.2.1 Features of Hadoop

Let us cite a few features of Hadoop:

1. It is optimized to handle massive quantities of structured, semi-structured, and unstructured data, using commodity hardware, that is, relatively inexpensive computers.
2. Hadoop has a shared nothing architecture.
3. It replicates its data across multiple computers so that if one goes down, the data can still be processed from another machine that stores its replica.
4. Hadoop is for high throughput rather than low latency. It is a batch operation handling massive quantities of data; therefore the response time is not immediate.
5. It complements On-Line Transaction Processing (OLTP) and On-Line Analytical Processing (OLAP). However, it is not a replacement for a relational database management system.
6. It is NOT good when work cannot be parallelized or when there are dependencies within the data.
7. It is NOT good for processing small files. It works best with huge data files and data sets.

4.2.2 Key Advantages of Hadoop

Refer Figure 4.9 for a quick look at the key advantages of Hadoop:

1. **Stores data in its native format:** Hadoop's data storage framework (HDFS – Hadoop Distributed File System) can store data in its native format. There is no structure that is imposed while keying in data or storing data. HDFS is pretty much schema-less. It is only later when the data needs to be processed that structure is imposed on the raw data.

its "rich query language", "fast in-place update", etc. The chapter will cover the CRUD (Create, Read, Update, and Delete) operations in detail.

To gain the maximum from the chapter, please attempt the Test Me exercises given at the end of the chapter.

6.1 WHAT IS MONGODB?

MongoDB is

1. Cross-platform.
2. Open source.
3. Non-relational.
4. Distributed.
5. NoSQL.
6. Document-oriented data store.

6.2 WHY MONGODB?

Few of the major challenges with traditional RDBMS are dealing with large volumes of data, rich variety of data – particularly unstructured data, and meeting up to the scale needs of enterprise data. The need is for a database that can scale out or scale horizontally to meet the scale requirements, has flexibility with respect to schema, is fault tolerant, is consistent and partition tolerant, and can be easily distributed over a multitude of nodes in a cluster. Refer Figure 6.1.

6.2.1 Using Java Script Object Notation (JSON)

JSON is extremely expressive. MongoDB actually does not use JSON but BSON (pronounced Bee Son) – it is Binary JSON. It is an open standard. It is used to store complex data structures.

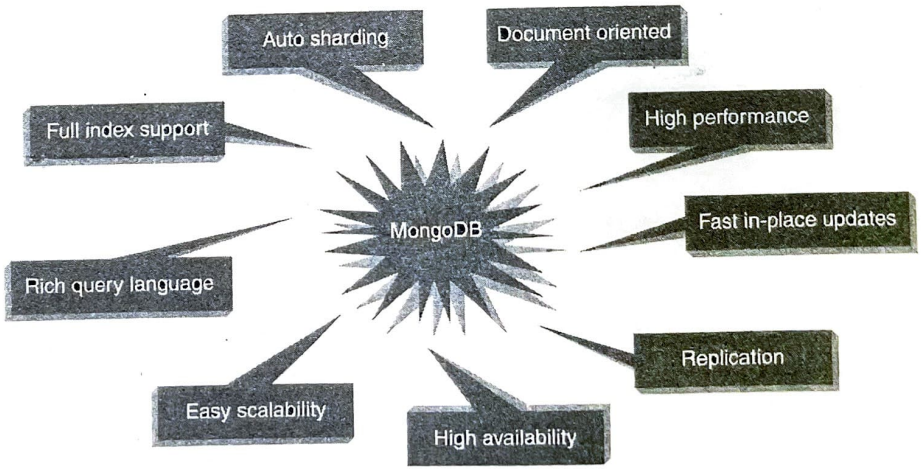


Figure 6.1 Why MongoDB?

Let us trace the journey from .csv to XML to JSON: Let us look at how data is stored in .csv file. Assume that this data is about the employees of an organization named "XYZ". As can be seen below, the column values are separated using commas and the rows are separated by a carriage return.

John, Mathews, +123 4567 8900
Andrews, Symmonds, +456 7890 1234
Mable, Mathews, +789 1234 5678

This looks good! However let us make it slightly more legible by adding column heading.

FirstName, LastName, ContactNo
John, Mathews, +123 4567 8900
Andrews, Symmonds, +456 7890 1234
Mable, Mathews, +789 1234 5678

Now assume that few employees have more than one ContactNo. It can be neatly classified as OfficeContactNo and HomeContactNo. But what if few employees have more than one OfficeContactNo and more than one HomeContactNo? Ok, so this is the first issue we need to address.

Let us look at just another piece of data that you wish to store about the employees. You need to store their email addresses as well. Here again we have the same issues, few employees have two email addresses, few have three and there are a few employees with more than three email addresses as well.

As we come across these fields or columns, we realize that it gets messy with .csv. CSV are known to store data well if it is flat and does not have repeating values.

The problem becomes even more complex when different departments maintain the details of their employees. The formats of .csv (columns, etc.) could vastly differ and it will call for some efforts before we can merge the files from the various departments to make a single file.

This problem can be solved by XML. But as the name suggests XML is highly extensible. It does not just call for defining a data format, rather it defines how you define a data format. You may be prepared to undertake this cumbersome task for highly complex and structured data; however, for simple data exchange it might just be too much work.

Enter JSON! Let us look at how it reacts to the problem at hand.

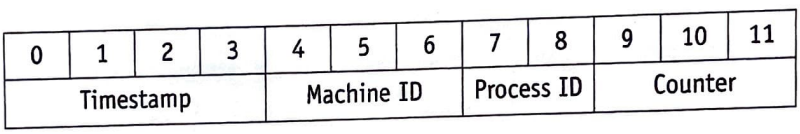
```
{
FirstName: John,
LastName: Mathews,
ContactNo: [+123 4567 8900, +123 4444 5555]
}
{
FirstName: Andrews,
LastName: Symmonds,
ContactNo: [+456 7890 1234, +456 6666 7777]
}
{
FirstName Mable,
LastName: Mathews,
ContactNo: +789 1234 5678
}
```

As you can see it is quite easy to read a JSON. There is absolutely no confusion now. One can have a list of *n* contact numbers, and they can be stored with ease.

JSON is very expressive. It provides the much needed ease to store and retrieve documents in their real form. The binary form of JSON is BSON. BSON is an open standard. In most cases it consumes less space as compared to the text-based JSON. There is yet another advantage with BSON. It is much easier and quicker to convert BSON to a programming language's native data format. There are MongoDB drivers available for a number of programming languages such as C, C++, Ruby, PHP, Python, C#, etc., and each works slightly differently. Using the basic binary format enables the native data structures to be built quickly for each language without going through the hassle of first processing JSON.

6.2.2 Creating or Generating a Unique Key

Each JSON document should have a unique identifier. It is the _id key. It is similar to the primary key in relational databases. This facilitates search for documents based on the unique identifier. An index is automatically built on the unique identifier. It is your choice to either provide unique values yourself or have the mongo shell generate the same.



6.2.2.1 *Database*

It is a collection of collections. In other words, it is like a container for collections. It gets created the first time that your collection makes a reference to it. This can also be created on demand. Each database gets its own set of files on the file system. A single MongoDB server can house several databases.

6.2.2.2 *Collection*

A collection is analogous to a table of RDBMS. A collection is created on demand. It gets created the first time that you attempt to save a document that references it. A collection exists within a single database. A collection holds several MongoDB documents. A collection does not enforce a schema. This implies that documents within a collection can have different fields. Even if the documents within a collection have same fields, the order of the fields can be different.

6.2.2.3 *Document*

A document is analogous to a row/record/tuple in an RDBMS table. A document has a dynamic schema. This implies that a document in a collection need not necessarily have the same set of fields/key-value pairs. Shown in Figure 6.2 is a collection by the name "students" containing three documents.

6.2.3 Support for Dynamic Queries

MongoDB has extensive support for dynamic queries. This is in keeping with traditional RDBMS wherein we have static data and dynamic queries. CouchDB, another document-oriented, schema-less NoSQL database and MongoDB's biggest competitor, works on quite the reverse philosophy. It has support for dynamic data and static queries.

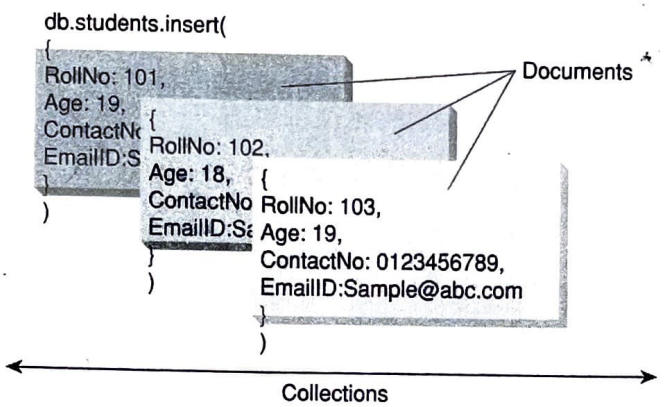


Figure 6.2 A collection "students" containing 3 documents.

6.2.4 Storing Binary Data

MongoDB provides GridFS to support the storage of binary data. It can store up to 4 MB of data. This usually suffices for photographs (such as a profile picture) or small audio clips. However, if one wishes to store movie clips, MongoDB has another solution.

It stores the metadata (data about data along with the context information) in a collection called "file". It then breaks the data into small pieces called chunks and stores it in the "chunks" collection. This process takes care about the need for easy scalability.

6.2.5 Replication

Why replication? It provides data redundancy and high availability. It helps to recover from hardware failure and service interruptions. In MongoDB, the replica set has a single primary and several secondaries. Each write request from the client is directed to the primary. The primary logs all write requests into its Oplog (operations log). The Oplog is then used by the secondary replica members to synchronize their data. This way there is strict adherence to consistency. Refer Figure 6.3. The clients usually read from the primary. However, the client can also specify a read preference that will then direct the read operations to the secondary.

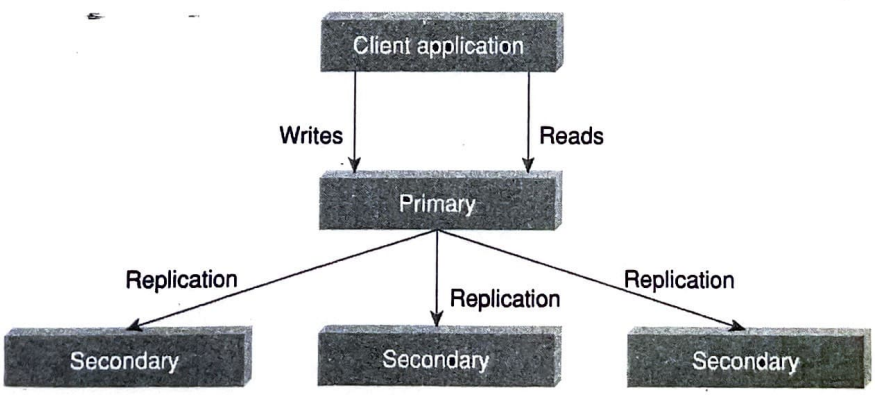


Figure 6.3 The process of **REPLICATION** in MongoDB.

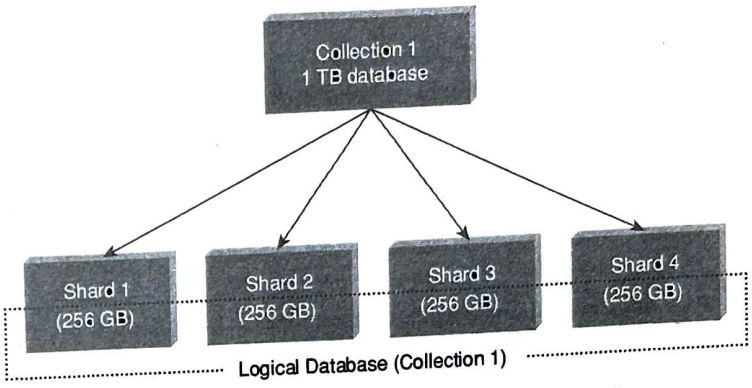


Figure 6.4 The process of **SHARDING** in MongoDB.

6.2.6 Sharding

Sharding is akin to horizontal scaling. It means that the large dataset is divided and distributed over multiple servers or shards. Each shard is an independent database and collectively they would constitute a logical database.

The prime advantages of sharding are as follows:

1. Sharding reduces the amount of data that each shard needs to store and manage. For example, if the dataset was 1 TB in size and we were to distribute this over four shards, each shard would house just 256 GB data. Refer Figure 6.4. As the cluster grows, the amount of data that each shard will store and manage will decrease.
2. Sharding reduces the number of operations that each shard handles. For example, if we were to insert data, the application needs to access only that shard which houses that data.

6.2.7 Updating Information In-Place

MongoDB updates the information in-place. This implies that it updates the data wherever it is available. It does not allocate separate space and the indexes remain unaltered.

MongoDB is all for lazy-writes. It writes to the disk once every second. Reading and writing to disk is a slow operation as compared to reading and writing from memory. The fewer the reads and writes that we perform to the disk, the better is the performance. This makes MongoDB faster than its other competitors who write almost immediately to the disk. However, there is a tradeoff. MongoDB makes no guarantee that data will be stored safely on the disk.

Guess Me

A. Who am I?

- I am blindingly fast
- I am massively scalable
- I am easy to use
- I work with documents rather than rows

B. Who am I?

- I am not for everyone
- I am good with complex data structures such as blog posts and comments
- I am good with analytics such as a real time google analytics
- I am comfortable with Linux, Mac OS, Solaris, and windows

C. Who am I?

- I have support for transactions
- I have static data
- I allow dynamic queries to be run on me

D. Who am I?

- I am one of the biggest competitor for MongoDB
- I have dynamic data
- Only static queries can be run on me
- I am document-oriented too

Answers:

A. MongoDB
B. MongoDB
C. Traditional RDBMS
D. CouchDB

6.3 TERMS USED IN RDBMS AND MONGODB

RDBMS	MongoDB
Database	Database
Table	Collection
Record	Document
Columns	Fields / Key Value pairs
Index	Index
Joins	Embedded documents
Primary Key	Primary key (_id is a identifier)

	MySQL	Oracle	MongoDB
Database Server	MySqld	Oracle	Mongod
Database Client	MySql	SQL Plus	mongo

6.3.1 Create Database

The syntax for creating database is as follows:

use DATABASE_Name

To create a database by the name "myDB" the syntax is

use myDB

```
> use myDB;
switched to db myDB
>
```

To confirm the existence of your database, type the command at the MongoDB shell:

db

```
> db;
myDB
>
```

To get a list of all databases, type the below command:

show dbs

```
> show dbs;
admin  (empty)
local  0.078GB
test   0.078GB
>
```

Notice that the newly created database, "myDB" does not show up in the list above. The reason is that the database needs to have at least one document to show up in the list.

The default database in MongoDB is test. If one does not create any database, all collections are by default stored in the test database.

6.3.2 Drop Database

The syntax to drop database is as follows:

db.dropDatabase();

To drop the database, "myDB", first ensure that you are currently placed in "myDB" database and then use the db.dropDatabase() command to drop the database.

use myDB;
db.dropDatabase();

Confirm if the database "myDB" has been dropped.

```
> db.dropDatabase();
{ "dropped" : "myDB", "ok" : 1 }
>
```

If no database is selected, the default database "test" is dropped.

6.4 DATA TYPES IN MONGODB

The following are various data types in MongoDB.

String	Must be UTF-8 valid. Most commonly used data type.
Integer	Can be 32-bit or 64-bit (depends on the server).
Boolean	To store a true/false value.
Double	To store floating point (real values).
Min/Max keys	To compare a value against the lowest or highest BSON elements.
Arrays	To store arrays or list or multiple values into one key.
Timestamp	To record when a document has been modified or added.
Null	To store a NULL value. A NULL is a missing or unknown value.
Date	To store the current date or time in Unix time format. One can create object of date and pass day, month and year to it.
Object ID	To store the document's id.
Binary data	To store binary data (images, binaries, etc.).
Code	To store javascript code into the document.
Regular expression	To store regular expression.

A few commands worth looking at are as follows (try them!!!).

To report the name of the current database:

```
C:\windows\system32\cmd.exe - mongo
> db
test
>
```

To display the list of databases:

```
C:\windows\system32\cmd.exe - mongo
> show dbs
admin  (empty)
local  0.078GB
myDB1  0.078GB
>
```

To switch to a new database, for example, myDB1:

```
C:\windows\system32\cmd.exe - mongo
> use myDB1
switched to db myDB1
>
```

To display the list of collections (tables) in the current database:

```
C:\windows\system32\cmd.exe - mongo
> show collections
system.indexes
system.js
>
```

To display the current version of the MongoDB server:

```
C:\windows\system32\cmd.exe - mongo
> db.version()
2.6.1
>
```

WHAT'S IN STORE?

This chapter will cover another NoSQL database called "Cassandra". We will explore the features of Cassandra that has made it so immensely popular. The chapter will cover the basic CRUD (Create, Read, Update, and Delete) operations using cqlsh.

Please attempt the Test Me exercises given at the end of the chapter to practice, learn, and comprehend Cassandra effectively.

7.1 APACHE CASSANDRA – AN INTRODUCTION

We shall start this chapter with few points that a reader should know about Cassandra.

1. Apache Cassandra was born at Facebook. After Facebook open sourced the code in 2008, Cassandra became an Apache Incubator project in 2009 and subsequently became a top-level Apache project in 2010.
2. It is built on Amazon's dynamo and Google's BigTable.
3. Cassandra does NOT compromise on availability. Since it does not have a master-slave architecture, there is no question of single point of failure. This proves beneficial for business critical applications that need to be up and running always and cannot afford to go down ever.
4. It is highly scalable (it scales out), high performance distributed database. It distributes and manages gigantic amount of data across commodity servers.

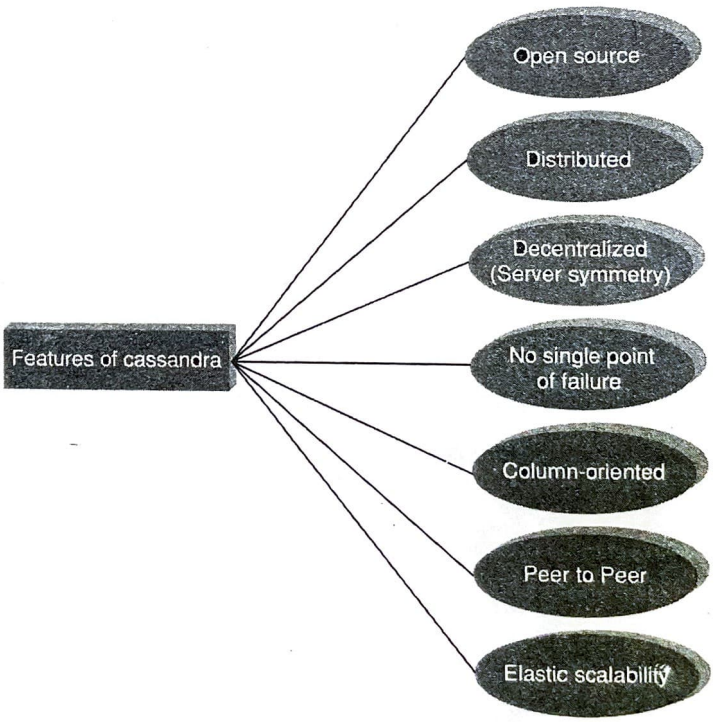


Figure 7.1 Features of Cassandra.

5. It is a column-oriented database designed to support peer-to-peer symmetric nodes instead of the master-slave architecture.
6. It has adherence to the Availability and Partition Tolerance properties of CAP theorem. It takes care of consistency using BASE (Basically Available Soft State Eventual Consistency) approach.

Refer Figure 7.1. Few companies that have successfully deployed Cassandra and have benefitted immensely from it are as follows:

1. Twitter
2. Netflix
3. Cisco
4. Adobe
5. eBay
6. Rackspace

7.2 FEATURES OF CASSANDRA

7.2.1 Peer-to-Peer Network

As with any other NoSQL database, Cassandra is designed to distribute and manage large data loads across multiple nodes in a cluster constituted of commodity hardware. Cassandra does NOT have a master-slave architecture which means that it does NOT have single point of failure. A node in Cassandra is structurally identical to any other node. Refer Figure 7.2. In case a node fails or is taken offline, it definitely impacts the throughput. However, it is a case of graceful degradation where everything does not come crashing at any given instant owing to a node failure. One can still go about business as usual. It tides over the problem of failure by employing a peer-to-peer distributed system across homogeneous nodes. It ensures that data is distributed across all nodes in the cluster. Each node exchanges information across the cluster every second.

Let us look at how a Cassandra node writes. Each write is written to the commit log sequentially. A write is taken to be successful only if it is written to the commit log. Data is then indexed and pushed to an in-memory structure called "Memtable". When the in-memory data structure, "the Memtable", is full, the contents are flushed to "SSTable" (Sorted String) data file on the disk. The SSTable is immutable and is

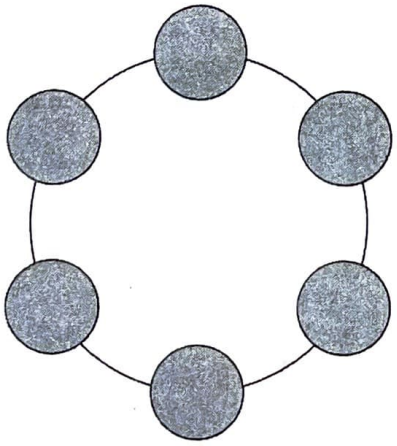


Figure 7.2 Sample Cassandra cluster.

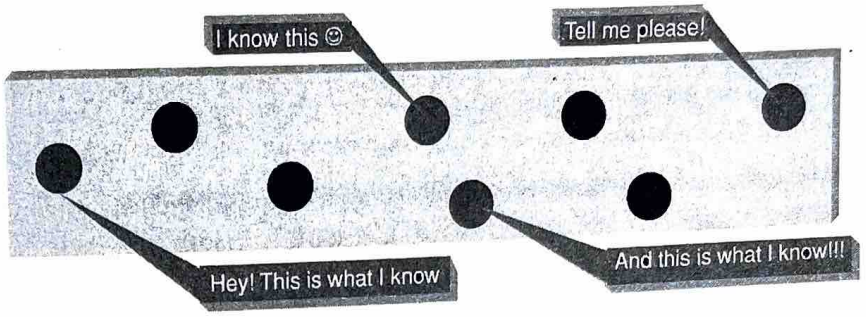


Figure 7.3 Gossip protocol.

append-only. It is stored on disk sequentially and is maintained for each Cassandra table. The partitioning and replication of all writes are performed automatically across the cluster.

7.2.2 Gossip and Failure Detection

Gossip protocol is used for intra-ring communication. It is a peer-to-peer communication protocol which eases the discovery and sharing of location and state information with other nodes in the cluster. Refer Figure 7.3. Although there are quite a few subtleties involved, but at its core it's a simple and robust system. A node only has to send out the communication to a subset of other nodes. For repairing unread data, Cassandra uses what's called an anti-entropy version of the gossip protocol.

7.2.3 Partitioner

A partitioner takes a call on how to distribute data on the various nodes in a cluster. It also determines the node on which to place the very first copy of the data. Basically a partitioner is a hash function to compute the token of the partition key. The partition key helps to identify a row uniquely.

7.2.4 Replication Factor

The replication factor determines the number of copies of data (replicas) that will be stored across nodes in a cluster. If one wishes to store only one copy of each row on one node, they should set the replication factor to one. However, if the need is for two copies of each row of data on two different nodes, one should go with a replication factor of two. The replication factor should ideally be more than one and not more than the number of nodes in the cluster. A replication strategy is employed to determine which nodes to place the data on. Two replication strategies are available:

1. SimpleStrategy.
2. NetworkToplogyStrategy.

The preferred one is NetworkTopologyStrategy as it is simple and supports easy expansion to multiple data centers, should there be a need.

7.2.5 Anti-Entropy and Read Repair

A cluster is made up of several nodes. Since the cluster is constituted of commodity hardware, it is prone to failure. In order to achieve fault tolerance, a given piece of data is replicated on one or more nodes. A client