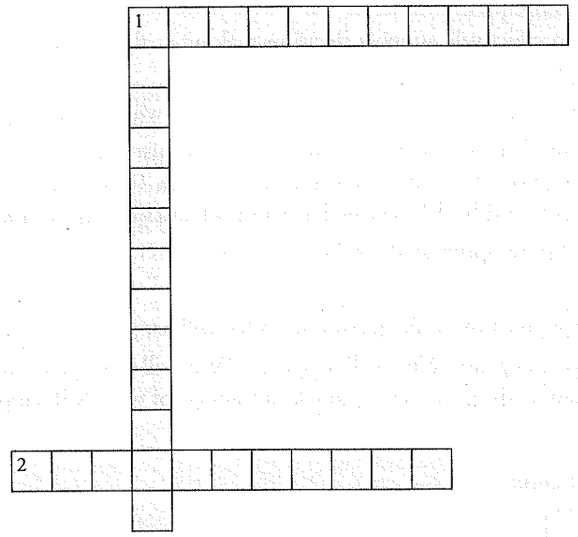
2. Puzzle on Architecture



Across

1. _____ is an important advantage of shared nothing architecture.
2. In this architecture, multiple processors have their own private memory.

Down

1. In this architecture, central memory is shared by multiple processors.

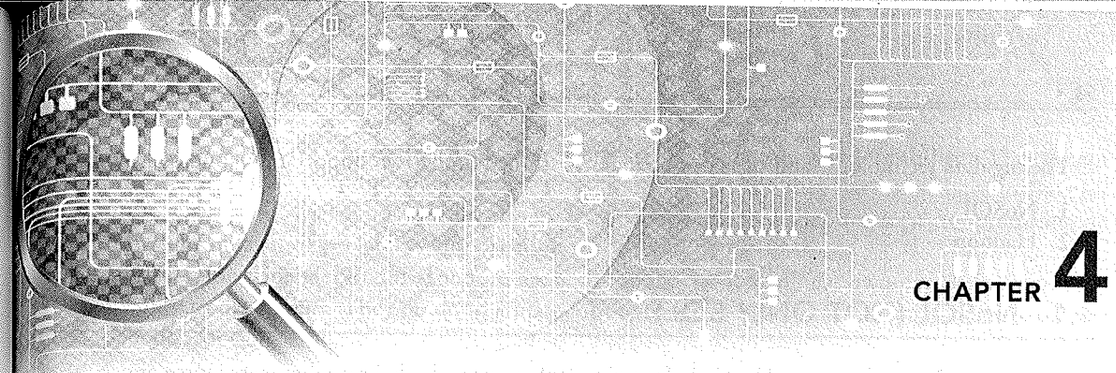
Answer:

Across

1. Scalability
2. Shared Disk

Down

1. Shared Memory



CHAPTER 4

The Big Data Technology Landscape

BRIEF CONTENTS

- What's in Store?
- NoSQL (Not Only SQL)
  - Where is it used?
  - What is it?
  - Types of NoSQL Databases
  - Why NoSQL?
  - Advantages of NoSQL
  - What we miss with NoSQL?
  - Use of NoSQL in Industry
  - NoSQL Vendors
  - SQL versus NoSQL
  - NewSQL
  - Comparison of SQL, NoSQL, and NewSQL
- Hadoop
  - Features of Hadoop
  - Key Advantages of Hadoop
  - Versions of Hadoop
    - Hadoop 1.0
    - Hadoop 2.0
- Overview of Hadoop Ecosystems
- Hadoop Distributions
- Hadoop versus SQL
- Integrated Hadoop Systems Offered by Leading Market Vendors
- Cloud-Based Hadoop Solutions

"The goal is to turn data into information, and information into insight."

– Carly Fiorina, former CEO, Hewlett-Packard Co

WHAT'S IN STORE?

The focus of this chapter is on understanding "big data technology landscape". This chapter is an overview on NoSQL and Hadoop. There are separate chapters on NoSQL (MongoDB and Cassandra) as well as Hadoop in the book.

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.

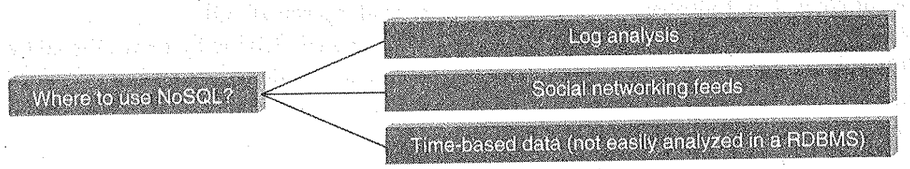


Figure 4.1 Where to use NoSQL?

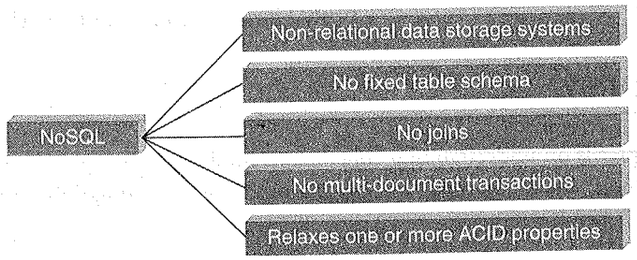


Figure 4.2 What is NoSQL?

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.

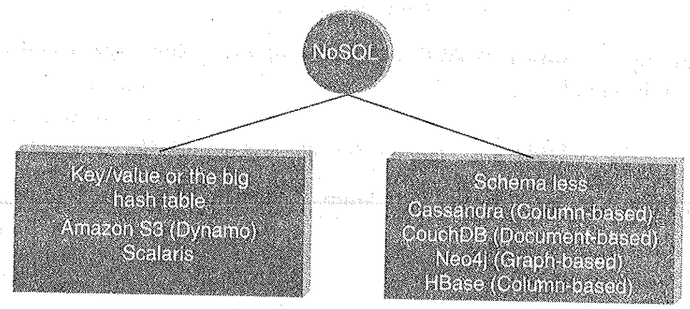
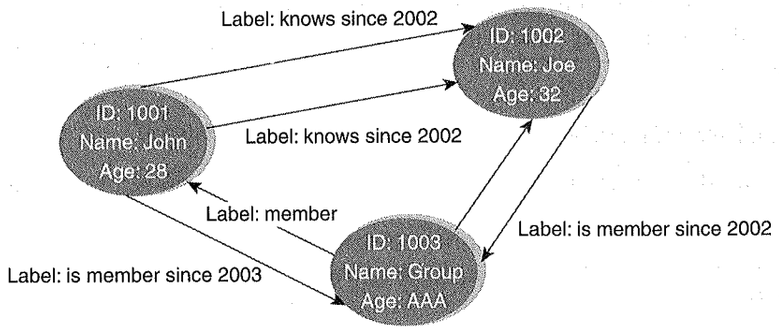


Figure 4.3 Types of NoSQL databases.

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

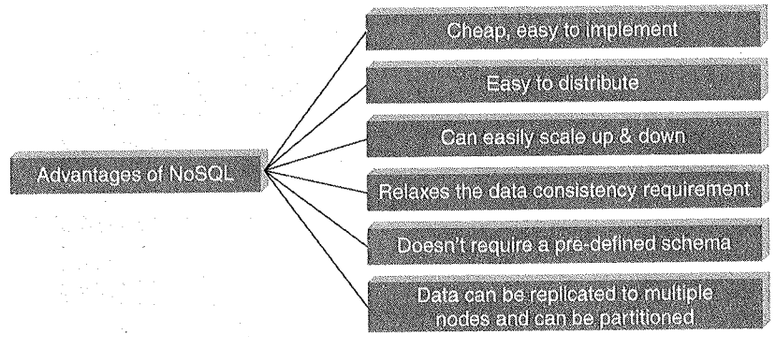


Figure 4.4 Advantages of NoSQL.

(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.

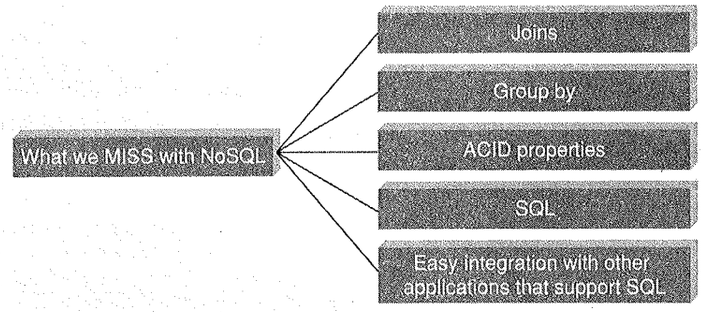


Figure 4.5 What we miss with NoSQL?

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.

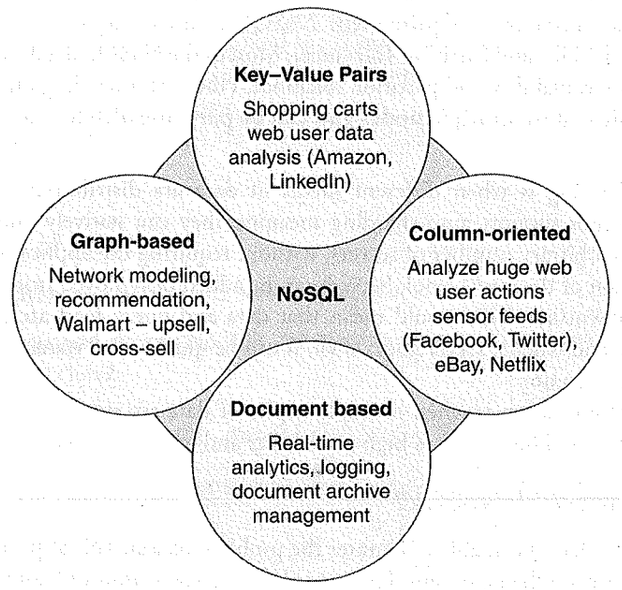


Figure 4.6 Use of NoSQL in industry.

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.

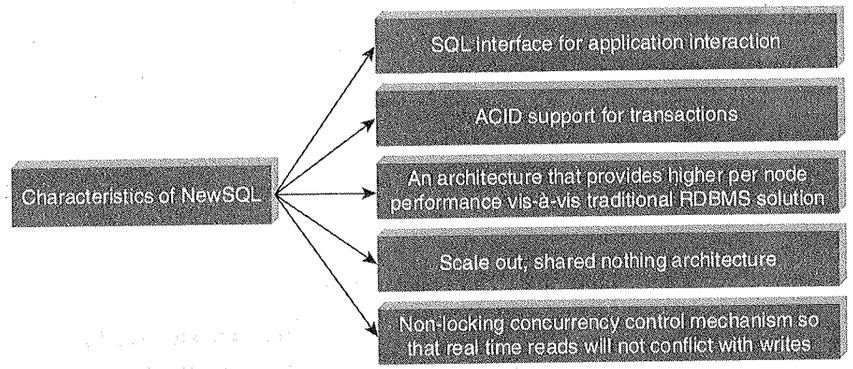


Figure 4.7 Characteristics of NewSQL.

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 Adherence to data model	Yes Adherence to relational model	No	Maybe
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

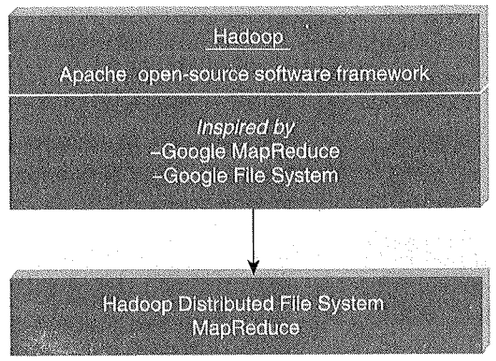


Figure 4.8 Hadoop.

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.

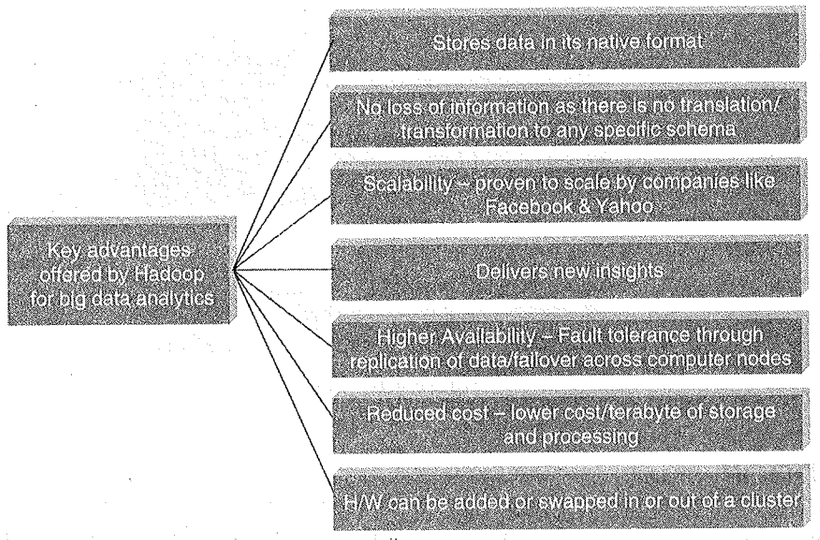


Figure 4.9 Key advantages of Hadoop.

2. **Scalable:** Hadoop can store and distribute very large datasets (involving thousands of terabytes of data) across hundreds of inexpensive servers that operate in parallel.
3. **Cost-effective:** Owing to its scale-out architecture, Hadoop has a much reduced cost/terabyte of storage and processing.
4. **Resilient to failure:** Hadoop is fault-tolerant. It practices replication of data diligently which means whenever data is sent to any node, the same data also gets replicated to other nodes in the cluster, thereby ensuring that in the event of a node failure, there will always be another copy of data available for use.
5. **Flexibility:** One of the key advantages of Hadoop is its ability to work with all kinds of data: structured, semi-structured, and unstructured data. It can help derive meaningful business insights from email conversations, social media data, click-stream data, etc. It can be put to several purposes such as log analysis, data mining, recommendation systems, market campaign analysis, etc.
6. **Fast:** the processing is extremely fast in Hadoop compared to other conventional systems owing to the "move code to data" paradigm.

4.2.3 Versions of Hadoop

There are two versions of Hadoop available:

1. Hadoop 1.0
2. Hadoop 2.0

Let us take a look at the features of both. Refer Figure 4.10.

4.2.3.1 Hadoop 1.0

It has two main parts:

1. **Data storage framework:** It is a general-purpose file system called Hadoop Distributed File System (HDFS). HDFS is schema-less. It simply stores data files and these data files can be in just about any

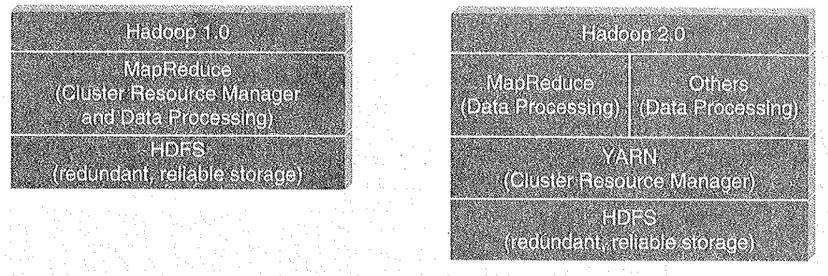


Figure 4.10 Versions of Hadoop.

format. The idea is to store files as close to their original form as possible. This is turn provides the business units and the organization the much needed flexibility and agility without being overly worried by what it can implement.

2. **Data processing framework:** This is a simple functional programming model initially popularized by Google as MapReduce. It essentially uses two functions: the MAP and the REDUCE functions to process data. The "Mappers" take in a set of key-value pairs and generate intermediate data (which is another list of key-value pairs). The "Reducers" then act on this input to produce the output data. The two functions seemingly work in isolation from one another, thus enabling the processing to be highly distributed in a highly-parallel, fault-tolerant, and scalable way.

There were however a few limitations of Hadoop 1.0. They are as follows:

1. The first limitation was the requirement for MapReduce programming expertise along with proficiency required in other programming languages, notably Java.
2. It supported only batch processing which although is suitable for tasks such as log analysis, large-scale data mining projects but pretty much unsuitable for other kinds of projects.
3. One major limitation was that Hadoop 1.0 was tightly computationally coupled with MapReduce, which meant that the established data management vendors were left with two options: Either rewrite their functionality in MapReduce so that it could be executed in Hadoop or extract the data from HDFS and process it outside of Hadoop. None of the options were viable as it led to process inefficiencies caused by the data being moved in and out of the Hadoop cluster.

Let us look at whether these limitations have been wholly or in parts resolved by Hadoop 2.0.

4.2.3.2 Hadoop 2.0

In Hadoop 2.0, HDFS continues to be the data storage framework. However, a new and separate resource management framework called **Y**et **A**nother **R**esource **N**egotiator (YARN) has been added. Any application capable of dividing itself into parallel tasks is supported by YARN. YARN coordinates the allocation of subtasks of the submitted application, thereby further enhancing the flexibility, scalability, and efficiency of the applications. It works by having an ApplicationMaster in place of the erstwhile JobTracker, running applications on resources governed by a new NodeManager (in place of the erstwhile TaskTracker). ApplicationMaster is able to run any application and not just MapReduce.

This, in other words, means that the MapReduce Programming expertise is no longer required. Furthermore, it not only supports batch processing but also real-time processing. MapReduce is no longer the only data processing option; other alternative data processing functions such as data standardization, master data management can now be performed natively in HDFS.

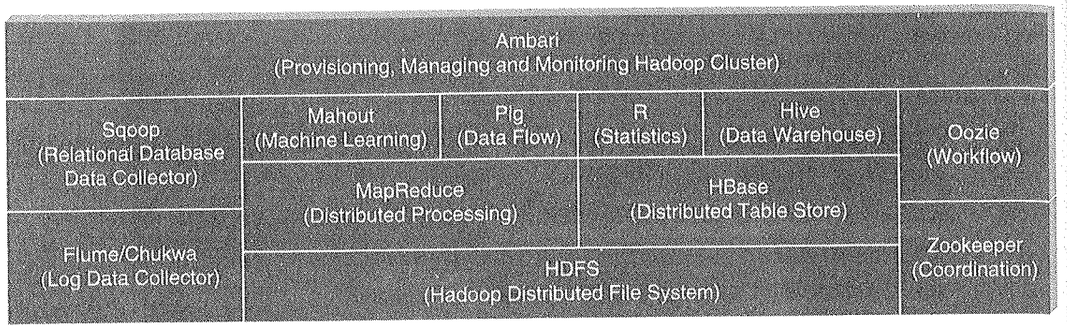


Figure 4.11 Hadoop ecosystem.

4.2.4 Overview of Hadoop Ecosystems

We will discuss the Hadoop ecosystem in brief here. It will be covered in detail in Chapter 5. The following are the components of the Hadoop ecosystem (shown in Figure 4.11):

1. **HDFS:** Hadoop Distributed File System. It simply stores data files as close to the original form as possible.
2. **HBase:** It is Hadoop's database and compares well with an RDBMS. It supports structured data storage for large tables.
3. **Hive:** It enables analysis of large data sets using a language very similar to standard ANSI SQL. This implies that anyone familiar with SQL should be able to access data stored on a Hadoop cluster.
4. **Pig:** Pig is an easy to understand data flow language. It helps with the analysis of large datasets which is quite the order with Hadoop. Even if one does not have the proficiency in MapReduce programming, the analysts and the persons entrusted with the task of comprehending data will still be able to analyze the data in a Hadoop cluster as the Pig scripts are automatically converted into MapReduce jobs by the Pig interpreter. MapReduce programming will be covered in detail in Chapter 8.
5. **ZooKeeper:** It is a coordination service for distributed applications.
6. **Oozie:** It is a workflow scheduler system to manage Apache Hadoop jobs.
7. ***Mahout*:** It is a scalable machine learning and data mining library.
8. **Chukwa:** It is a data collection system for managing large distributed systems.
9. **Sqoop:** It is used to transfer bulk data between Hadoop and structured data stores such as relational databases.
10. **Ambari:** It is a web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters.

4.2.5 Hadoop Distributions

Hadoop is an open-source Apache project. Anyone can freely download the core aspects of Hadoop. The core aspects of Hadoop include the following:

1. Hadoop Common
2. **H**adoop **D**istributed **F**ile **S**ystem (HDFS)
3. Hadoop YARN (**Y**et **A**nother **R**esource **N**egotiator)
4. Hadoop MapReduce

There are few companies such as IBM, Amazon Web Services, Microsoft, Teradata, Hortonworks, Cloudera, etc. that have packaged Hadoop into a more easily consumable distributions or services. Although each of these companies have a slightly different strategy, the key essence remains its ability to distribute data and workloads across potentially thousands of servers thus making big data manageable data. A few Hadoop distributions are given in Figure 4.12.

4.2.6 Hadoop versus SQL

Table 4.5 lists the differences between Hadoop and SQL.

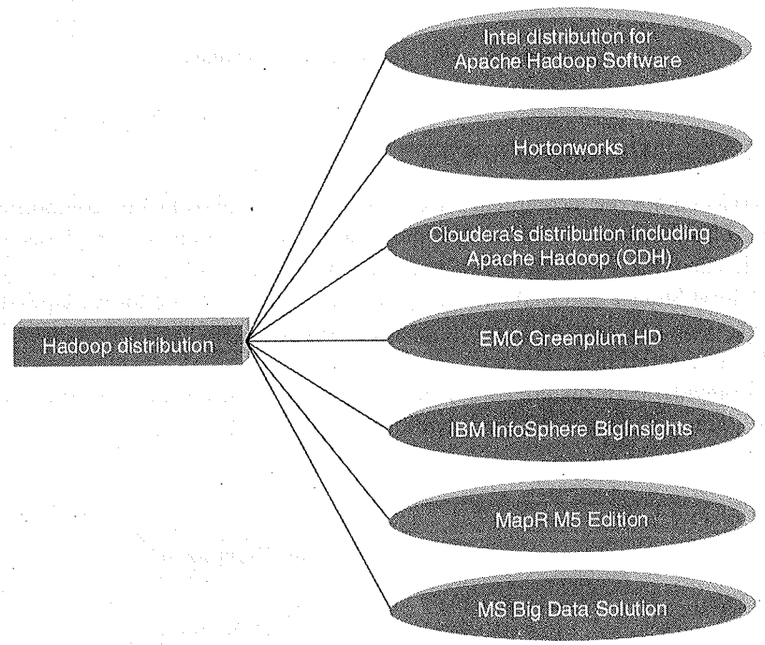


Figure 4.12 Hadoop distributions.

Table 4.5 Hadoop versus SQL

Hadoop	SQL
Scale out	Scale up
Key-Value pairs	Relational table
Functional Programming	Declarative Queries
Off-line batch processing	On-line transaction processing

4.2.7 Integrated Hadoop Systems Offered by Leading Market Vendors

Refer Figure 4.13 to get a glimpse of the leading market vendors offering integrated Hadoop systems.

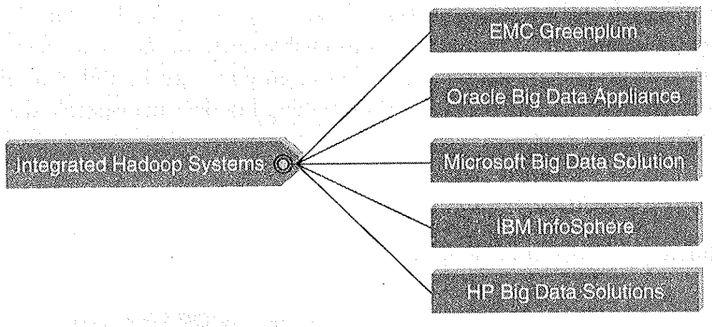


Figure 4.13 Integrated Hadoop systems.

4.2.8 Cloud-Based Hadoop Solutions

Amazon Web Services holds out a comprehensive, end-to-end portfolio of cloud computing services to help manage big data. The aim is to achieve this and more along with retaining the emphasis on reducing costs, scaling to meet demand, and accelerating the speed of innovation.

The Google Cloud Storage connector for Hadoop empowers one to perform MapReduce jobs directly on data in Google Cloud Storage, without the need to copy it to local disk and running it in the Hadoop Distributed File System (HDFS). The connector simplifies Hadoop deployment, and at the same time reduces cost and provides performance comparable to HDFS, all this while increasing reliability by eliminating the single point of failure of the name node. Refer Figure 4.14.

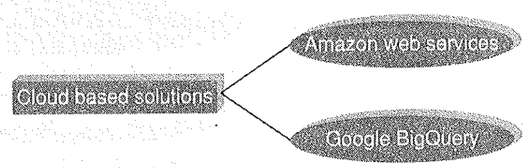


Figure 4.14 Cloud-based solutions.

REMIND ME

- NoSQL databases are non-relational, open source, distributed databases.
- NoSQL database allows insertion of data without a pre-defined schema.
- Hadoop has a shared nothing architecture.

- Hadoop 1.0 has two main parts:
  - Data storage framework
  - Data processing framework
- In Hadoop 2.0, a new and separate resource management framework called **Y**et **A**nother **R**esource **N**egotiator (YARN) has been added.

POINT ME (BOOKS)

- Hadoop for Dummies, Dirk Deroos, Paul C. Zikopoulos, Roman B. Melnyk, Bruce Brown, Wiley India Pvt. Ltd.
- NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Pramod J. Sadalage and Martin Fowler.

CONNECT ME (INTERNET RESOURCES)

- http://www.mongodb.com/nosql-explained
- http://nosql-database.org/
- http://www.techrepublic.com/blog/10-things/10-things-you-should-know-about-nosql-databases/
- http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduce_Compatibility_Hadoop1_Hadoop2.html
- http://hadoop.apache.org/

TEST ME

A. Fill Me

1. The expansion for CAP is _____, _____ and _____.
2. The expansion of BASE is _____.
3. MongoDB is _____ and _____.
4. Cassandra is _____ and _____.
5. _____ has no support for ACID properties of transactions.
6. _____ is a robust database that supports ACID properties of transactions and has the scalability of NoSQL.

Answers:

1. Consistency, Availability and Partition Tolerance
2. Basically Available Soft State Eventual Consistency
3. Consistent and Partition Tolerant
4. Available and Partition Tolerant
5. NoSQL
6. NewSQL

B. Place it in the Basket

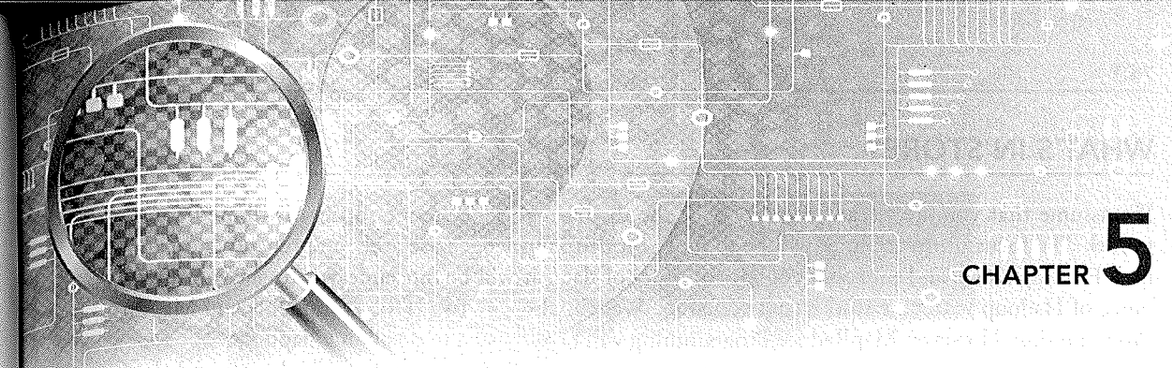
SQL	NoSQL

Following words are to be placed in the relevant basket:

(a) Relational
(b) Distributed
(c) Predefined schema
(d) Wide-column stores
(e) Vertically scalable
(f) Key–value pairs
(g) MySQL
(h) CouchDB
(i) Neo4j
(j) Cassandra
(k) Large dataset
(l) ACID properties
(m) Brewers CAP theorem
(n) Document based database
(o) Scales horizontally
(p) Avoids join operations
(q) JSON data
(r) Table or relations

Answers:

SQL	NoSQL
Relational	Distributed
Predefined schema	Wide-column stores
Vertically scalable	Key–value pairs
MySQL	CouchDB
ACID properties	Neo4j
Table or relations	Cassandra
	Large dataset
	Brewers CAP theorem
	Document based database
	Scales horizontally
	Avoids join operations
	JSON data



CHAPTER 5

Introduction to Hadoop

BRIEF CONTENTS

- What's in Store?
- Introducing Hadoop
  - Data: The Treasure Trove
- Why Hadoop?
- Why not RDBMS?
- RDBMS versus Hadoop
- Distributed Computing Challenges
  - Hardware Failure
  - How to Process this Gigantic Store of Data?
- History of Hadoop
  - The Name "Hadoop"
- Hadoop Overview
  - Key Aspects of Hadoop
  - Hadoop Components
  - Hadoop Conceptual Layer
  - High-Level Architecture of Hadoop
- Use Case for Hadoop
  - ClickStream Data
- Hadoop Distributors
- HDFS
  - HDFS Daemons
  - Anatomy of File Read
  - Anatomy of File Write
  - Replica Placement Strategy
  - Working with HDFS Commands
  - Special Features of HDFS
- Processing Data with Hadoop
  - MapReduce Daemons
  - How does MapReduce Work?
  - MapReduce Example
- Managing Resources and Applications with Hadoop YARN
  - Limitations of Hadoop 1.0 Architecture
  - HDFS Limitation
  - Hadoop 2: HDFS
  - Hadoop 2 YARN: Taking Hadoop Beyond Batch
- Interacting with Hadoop Ecosystem
  - Pig
  - Hive
  - Sqoop
  - HBase

"There were 5 exabytes of information created between the dawns of civilization through 2003, but that much information is now created every 2 days."

– Eric Schmidt, of Google, said in 2010

WHAT'S IN STORE?

We assume that you are already familiar with the distributed file system and the distributed computing model. The focus of this chapter will be to build on this knowledge base and comprehend and appreciate how Hadoop stores and processes colossal volumes of data. It will be our endeavor to get you the importance of Hadoop with case studies and scenarios. We will also discuss HDFS commands and MapReduce Programming. However, MapReduce Programming will be discussed in detail in Chapter 8.

We suggest you refer to some of the learning resources provided at the end of this chapter and also complete the "Test Me" exercises.

5.1 INTRODUCING HADOOP

Today, Big Data seems to be the buzz word! Enterprises, the world over, are beginning to realize that there is a huge volume of untapped information before them in the form of structured, semi-structured, and unstructured data. This varied variety of data is spread across the networks.

Let us look at few statistics to get an idea of the amount of data which gets generated every day, every minute, and every second.

1. **Every day:**
   (a) NYSE (New York Stock Exchange) generates 1.5 billion shares and trade data.
   (b) Facebook stores 2.7 billion comments and Likes.
   (c) Google processes about 24 petabytes of data.
2. **Every minute:**
   (a) Facebook users share nearly 2.5 million pieces of content.
   (b) Twitter users tweet nearly 300,000 times.
   (c) Instagram users post nearly 220,000 new photos.
   (d) YouTube users upload 72 hours of new video content.
   (e) Apple users download nearly 50,000 apps.
   (f) Email users send over 200 million messages.
   (g) Amazon generates over $80,000 in online sales.
   (h) Google receives over 4 million search queries.
3. **Every second:**
   (a) Banking applications process more than 10,000 credit card transactions.

5.1.1 Data: The Treasure Trove

1. Provides business advantages such as generating product recommendations, inventing new products, analyzing the market, and many, many more, ...
2. Provides few early key indicators that can turn the fortune of business.
3. Provides room for precise analysis. If we have more data for analysis, then we have greater precision of analysis.

To process, analyze, and make sense of these different kinds of data, we need a system that scales and addresses the challenges shown in Figure 5.1.

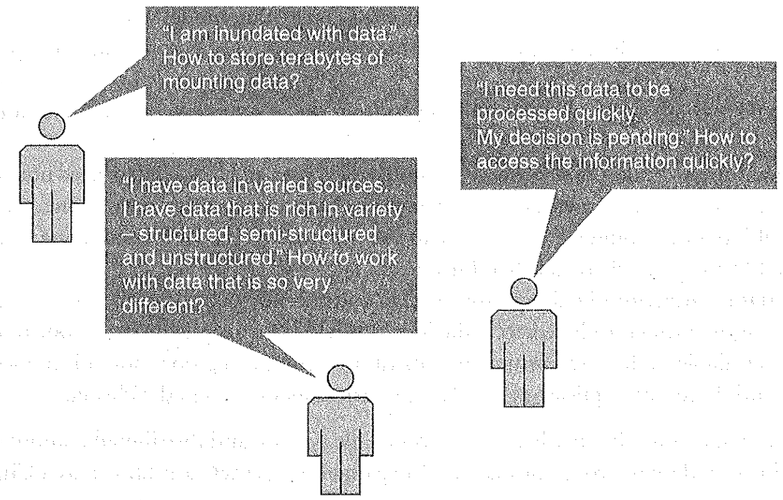


Figure 5.1 Challenges with big volume, variety, and velocity of data.

5.2 WHY HADOOP?

Ever wondered why Hadoop has been and is one of the most wanted technologies!!

The key consideration (the rationale behind its huge popularity) is:

Its capability to handle massive amounts of data, different categories of data – fairly quickly.

The other considerations are (Figure 5.2):

1. **Low cost:** Hadoop is an open-source framework and uses commodity hardware (commodity hardware is relatively inexpensive and easy to obtain hardware) to store enormous quantities of data.

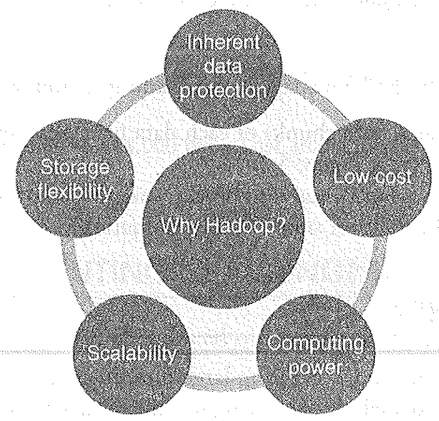


Figure 5.2 Key considerations of Hadoop.

2. **Computing power:** Hadoop is based on distributed computing model which processes very large volumes of data fairly quickly. The more the number of computing nodes, the more the processing power at hand.
3. **Scalability:** This boils down to simply adding nodes as the system grows and requires much less administration.
4. **Storage flexibility:** Unlike the traditional relational databases, in Hadoop data need not be pre-processed before storing it. Hadoop provides the convenience of storing as much data as one needs and also the added flexibility of deciding later as to how to use the stored data. In Hadoop, one can store unstructured data like images, videos, and free-form text.
5. **Inherent data protection:** Hadoop protects data and executing applications against hardware failure. If a node fails, it automatically redirects the jobs that had been assigned to this node to the other functional and available nodes and ensures that distributed computing does not fail. It goes a step further to store multiple copies (replicas) of the data on various nodes across the cluster.

Hadoop makes use of commodity hardware, distributed file system, and distributed computing as shown in Figure 5.3. In this new design, groups of machine are gathered together; it is known as a **Cluster**.

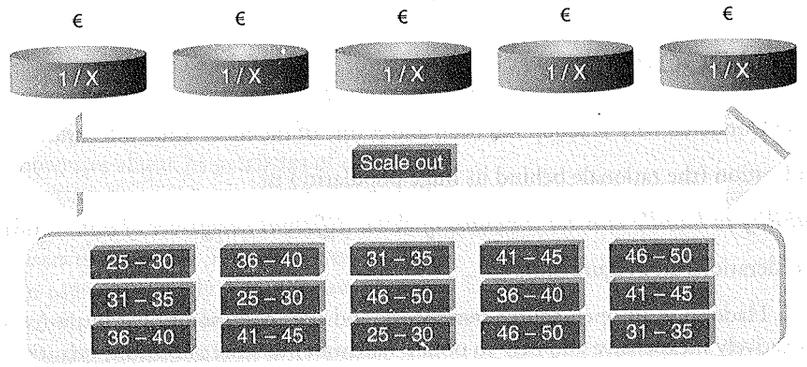


Figure 5.3 Hadoop framework (distributed file system, commodity hardware).

With this new paradigm, the data can be managed with **Hadoop** as follows:

1. Distributes the data and duplicates chunks of each data file across several nodes, for example, 25–30 is one chunk of data as shown in Figure 5.3.
2. Locally available compute resource is used to process each chunk of data in parallel.
3. Hadoop Framework handles failover smartly and automatically.

5.3 WHY NOT RDBMS?

RDBMS is not suitable for storing and processing large files, images, and videos. RDBMS is not a good choice when it comes to advanced analytics involving machine learning. Figure 5.4 describes the RDBMS system with respect to cost and storage. It calls for huge investment as the volume of data shows an upward trend.

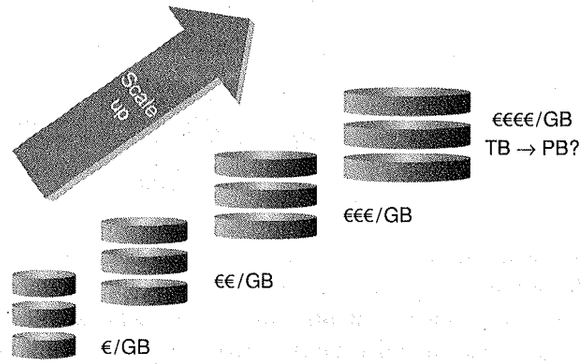


Figure 5.4 RDBMS with respect to cost/GB of storage.

5.4 RDBMS versus HADOOP

Table 5.1 describes the difference between RDBMS and Hadoop.

Table 5.1 RDBMS versus Hadoop

PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node Based Flat Structure.
Data	Suitable for structured data.	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Choice	When the data needs consistent relationship.	Big Data processing, which does not require any consistent relationships between data.
Processor	Needs expensive hardware or high-end processors to store huge volumes of data.	In a Hadoop Cluster, a node requires only a processor, a network card, and few hard drives.
Cost	Cost around $10,000 to $14,000 per terabytes of storage.	Cost around $4,000 per terabytes of storage.

5.5 DISTRIBUTED COMPUTING CHALLENGES

Although there are several challenges with distributed computing, we will focus on two major challenges.

5.5.1 Hardware Failure

In a distributed system, several servers are networked together. This implies that more often than not, there may be a possibility of hardware failure. And when such a failure does happen, how does one retrieve the

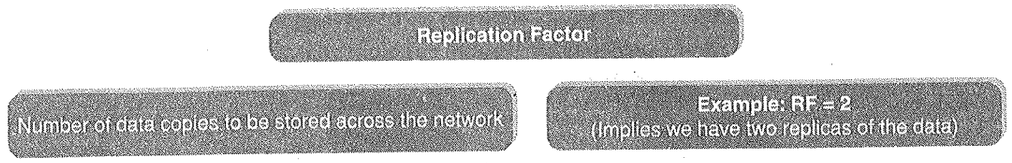


Figure 5.5 Replication factor.

data that was stored in the system? Just to explain further – a regular hard disk may fail once in 3 years. And when you have 1000 such hard disks, there is a possibility of at least a few being down every day.

Hadoop has an answer to this problem in **Replication Factor (RF)**. **Replication Factor** connotes the number of data copies of a given data item/data block stored across the network. Refer Figure 5.5.

JUST TO UNDERSTAND REPLICATION FURTHER, PICTURE THIS...

You work in a project team. There are six other members in the team. Each time there is an update related to the project work or an input received from the client, the project manager, Alex, ensures that he keeps at least three team members aware of the developments. You have been wondering at this style of working of your project manager. One day during the coffee break, when the project manager joins for coffee, you hesitantly ask him the question. Alex, "I had this question for you. Why is that each time we have an input from the client or any important piece of information, you leave it with at least three of our team members?" Alex smiled as he answered, "The reason is very simple. Assume that the client called and suggested some modification to the project. I shared it with just one person, let us say, person X. Tomorrow, when the suggested changes have to be incorporated, person X calls in sick. He is indisposed and not in office. Will that lead to our project coming to a standstill? Yes, isn't it? Therefore I share it with at least three team members, so that even if one is on leave or out of office for some reason, our work will not be stalled."

5.5.2 How to Process This Gigantic Store of Data?

In a distributed system, the data is spread across the network on several machines. A key challenge here is to integrate the data available on several machines prior to processing it.

Hadoop solves this problem by using **MapReduce** Programming. It is a programming model to process the data (MapReduce programming will be discussed a little later).

5.6 HISTORY OF HADOOP

Hadoop was created by Doug Cutting, the creator of Apache Lucene (a commonly used text search library). Hadoop is a part of the Apache Nutch (Yahoo) project (an open-source web search engine) and also a part of the Lucene project. Refer Figure 5.6 for more details.

5.6.1 The Name "Hadoop"

The name Hadoop is not an acronym; it's a made-up name. The project creator, Doug Cutting, explains how the name came about: "*The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term*".

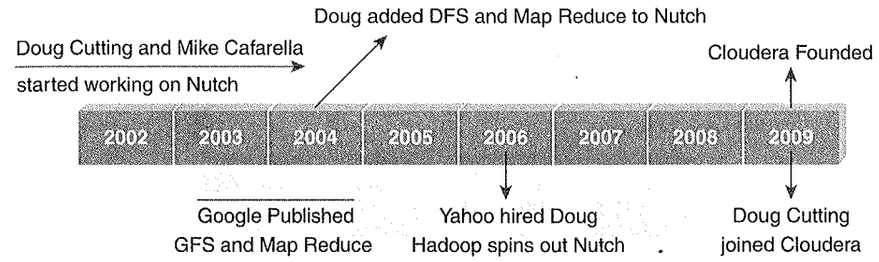


Figure 5.6 Hadoop history.

Subprojects and "contrib" modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig", for example).

Reference: Hadoop, The Definitive Guide, 3rd Edition, O'Reilly Publication Page. No. 9.

5.7 HADOOP OVERVIEW

Open-source software framework to store and process massive amounts of data in a distributed fashion on large clusters of commodity hardware. Basically, Hadoop accomplishes two tasks:

1. Massive data storage.
2. Faster data processing.

5.7.1 Key Aspects of Hadoop

Figure 5.7 describes the key aspects of Hadoop.

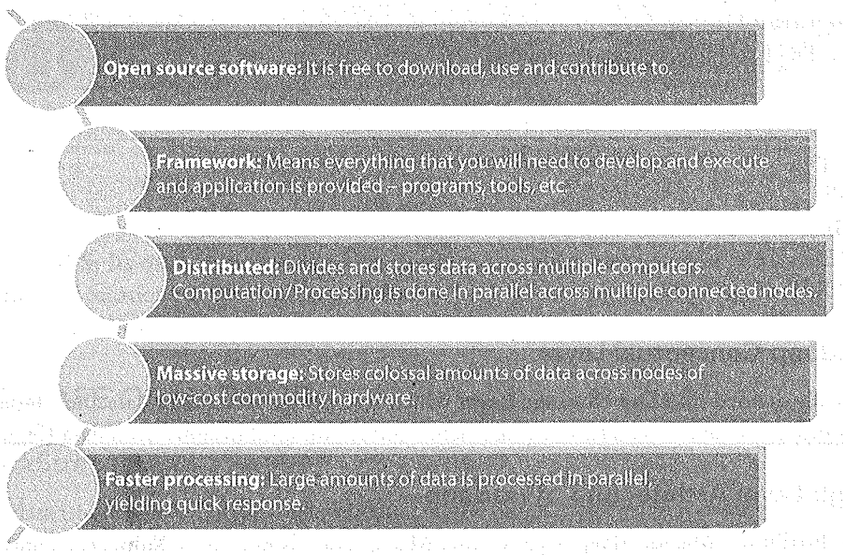


Figure 5.7 Key aspects of Hadoop.

5.7.2 Hadoop Components

Figure 5.8 depicts the Hadoop components.

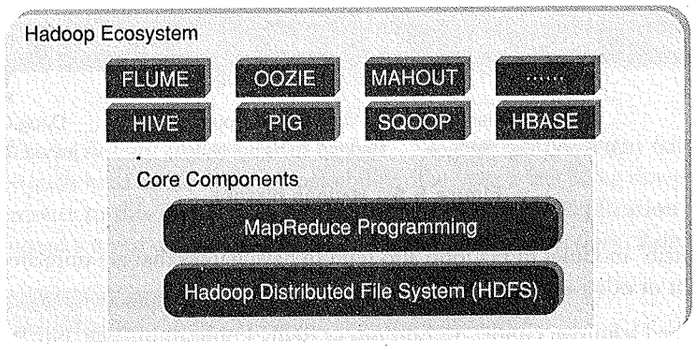


Figure 5.8 Hadoop components.

Hadoop Core Components

1. **HDFS:**
   (a) Storage component.
   (b) Distributes data across several nodes.
   (c) Natively redundant.
2. **MapReduce:**
   (a) Computational framework.
   (b) Splits a task across multiple nodes.
   (c) Processes data in parallel.

Hadoop Ecosystem: Hadoop Ecosystem are support projects to enhance the functionality of Hadoop Core Components. The Eco Projects are as follows:

1. HIVE
2. PIG
3. SQOOP
4. HBASE
5. FLUME
6. OOZIE
7. MAHOUT

5.7.3 Hadoop Conceptual Layer

It is conceptually divided into **Data Storage Layer** which stores huge volumes of data and **Data Processing Layer** which processes data in parallel to extract richer and meaningful insights from data (Figure 5.9).

5.7.4 High-Level Architecture of Hadoop

Hadoop is a distributed **Master-Slave** Architecture. Master node is known as **NameNode** and slave nodes are known as **DataNodes.** Figure 5.10 depicts the Master–Slave Architecture of Hadoop Framework.

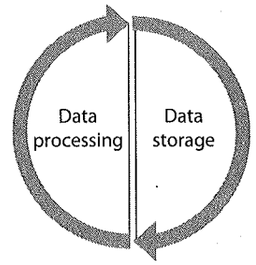


Figure 5.9 Hadoop conceptual layer.

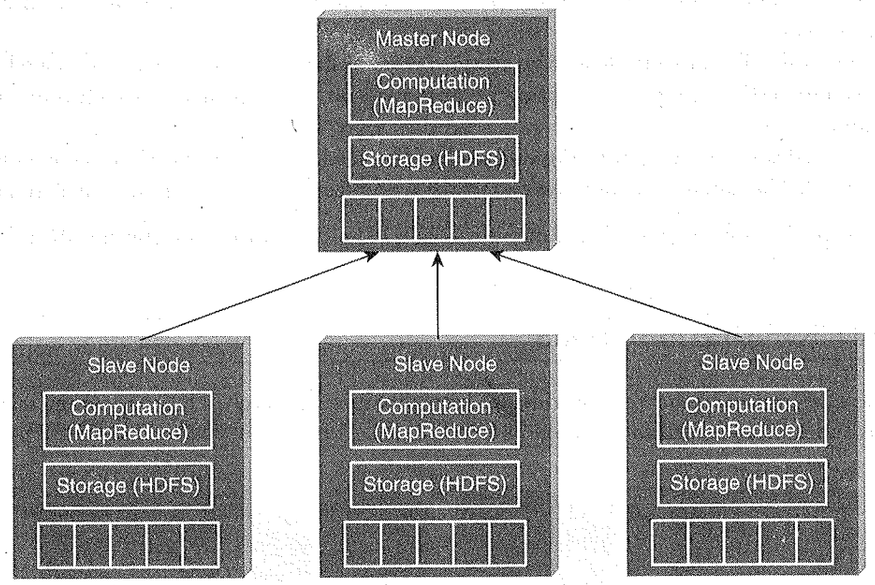


Figure 5.10 Hadoop high-level architecture.
Reference: Hadoop in Practice, Alex Holmes.

Let us look at the key components of the Master Node.

1. **Master HDFS:** Its main responsibility is partitioning the data storage across the slave nodes. It also keeps track of locations of data on DataNodes.
2. **Master MapReduce:** It decides and schedules computation task on slave nodes.

5.8 USE CASE OF HADOOP

5.8.1 ClickStream Data

ClickStream data (mouse clicks) helps you to understand the purchasing behavior of customers. ClickStream analysis helps online marketers to optimize their product web pages, promotional content, etc. to improve their business.

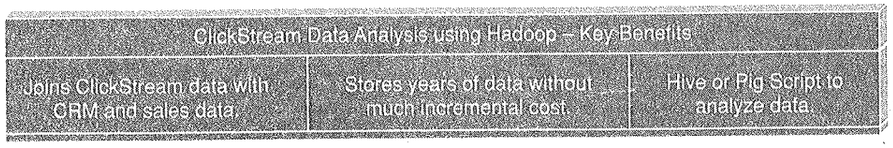


Figure 5.11 ClickStream data analysis.

The ClickStream analysis (Figure 5.11) using Hadoop provides **three key benefits**:

1. Hadoop helps to join ClickStream data with other data sources such as Customer Relationship Management Data (Customer Demographics Data, Sales Data, and Information on Advertising Campaigns). This additional data often provides the much needed information to understand customer behavior.
2. Hadoop's scalability property helps you to store years of data without ample incremental cost. This helps you to perform temporal or year over year analysis on ClickStream data which your competitors may miss.
3. Business analysts can use **Apache Pig** or **Apache Hive** for website analysis. With these tools, you can organize ClickStream data by user session, refine it, and feed it to visualization or analytics tools.

Reference: http://hortonworks.com/wp-content/uploads/2014/05/Hortonworks.BusinessValueofHadoop.v1.0.pdf

5.9 HADOOP DISTRIBUTORS

The companies shown in Figure 5.12 provide products that include Apache Hadoop, commercial support, and/or tools and utilities related to Hadoop.

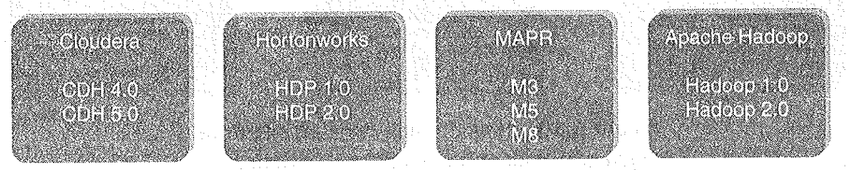


Figure 5.12 Common Hadoop distributors.

5.10 HDFS (HADOOP DISTRIBUTED FILE SYSTEM)

Some key Points of Hadoop Distributed File System are as follows:

1. Storage component of Hadoop.
2. Distributed File System.
3. Modeled after Google File System.
4. Optimized for high throughput (HDFS leverages large block size and moves computation where data is stored).
5. You can replicate a file for a configured number of times, which is tolerant in terms of both software and hardware.

6. Re-replicates data blocks automatically on nodes that have failed.
7. You can realize the power of HDFS when you perform read or write on large files (gigabytes and larger).
8. Sits on top of native file system such as ext3 and ext4, which is described in Figure 5.13.

Figure 5.14 describes important key points of HDFS. Figure 5.15 describes Hadoop Distributed File System Architecture. Client Application interacts with NameNode for metadata related activities and communicates with DataNodes to read and write files. DataNodes converse with each other for pipeline reads and writes.

Let us assume that the file "Sample.txt" is of size **192 MB**. As per the default data block size (64 MB), it will be split into three blocks and replicated across the nodes on the cluster based on the default replication factor.

5.10.1 HDFS Daemons

5.10.1.1 NameNode

HDFS breaks a large file into smaller pieces called **blocks**. NameNode uses a **rack ID** to identify DataNodes in the rack. A rack is a collection of DataNodes within the cluster. NameNode keeps tracks of blocks of a file as it is placed on various DataNodes. NameNode manages file-related operations such as read, write, create, and delete. Its main job is managing the **File System Namespace**. A file system namespace is collection of files in the cluster. NameNode stores HDFS namespace. File system namespace includes mapping of blocks to file, file properties and is stored in a file called **FsImage**. NameNode uses an EditLog (transaction log) to record every transaction that happens to the file system metadata. Refer Figure 5.16. When NameNode starts up, it reads FsImage and EditLog from disk and applies all transactions from the EditLog to in-memory representation of the FsImage. Then it flushes out new version of FsImage on disk and truncates the old EditLog because the changes are updated in the FsImage. There is a single NameNode per cluster.

Reference: http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html

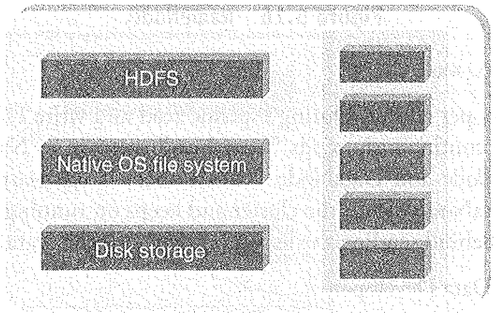


Figure 5.13 Hadoop Distributed File System.

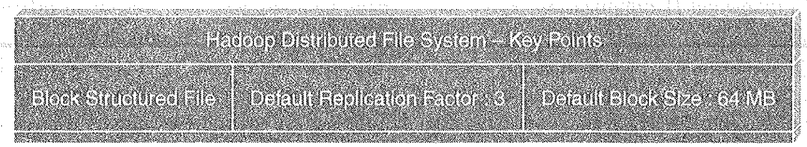


Figure 5.14 Hadoop Distributed File System – key points.

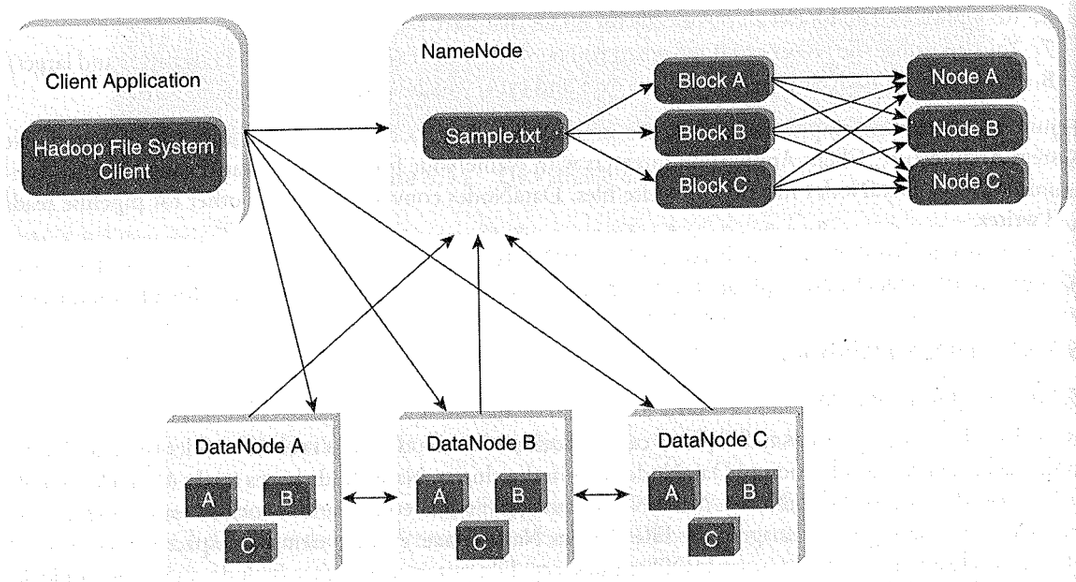


Figure 5.15 Hadoop Distributed File System Architecture.
Reference: Hadoop in Practice, Alex Holmes.

NameNode – Manages File Related Operations	
FsImage – File, in which entire file system is stored.	EditLog – Records every transaction that occurs to file system metadata.

Figure 5.16 NameNode.

5.10.1.2 DataNode

There are multiple DataNodes per cluster. During Pipeline read and write DataNodes communicate with each other. A DataNode also continuously sends **"heartbeat"** message to NameNode to ensure the connectivity between the NameNode and DataNode. In case there is no heartbeat from a DataNode, the NameNode replicates that DataNode within the cluster and keeps on running as if nothing had happened.

Let us explain the concept behind sending the heartbeat report by the DataNodes to the NameNode.

Reference: Wrox Certified Big Data Developer.

PICTURE THIS...

You work for a renowned IT organization. Every day when you come to office, you are required to swipe in to record your attendance. This record of attendance is then shared with your manager to keep him posted on who all from his team have reported for work. Your manager is able to allocate tasks to the team members who are present in office. The tasks for the day cannot be allocated to team members who have not turned in. Likewise heartbeat report is a way by which DataNodes inform the NameNode that they are up and functional and can be assigned tasks. Figure 5.17 depicts the above scenario.

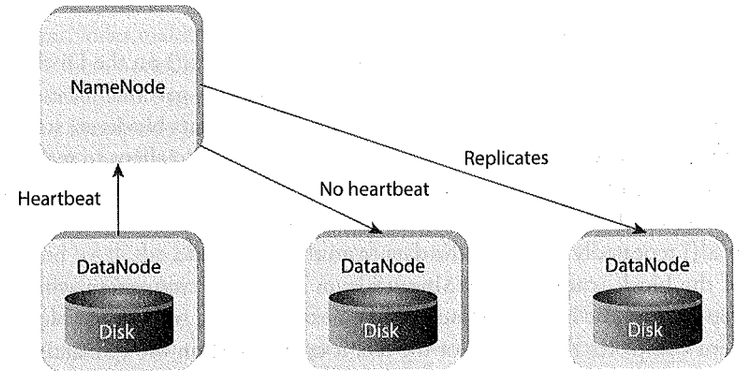


Figure 5.17 NameNode and DataNode Communication.

5.10.1.3 Secondary NameNode

The Secondary NameNode takes a snapshot of HDFS metadata at intervals specified in the Hadoop configuration. Since the memory requirements of Secondary NameNode are the same as NameNode, it is better to run NameNode and Secondary NameNode on different machines. In case of failure of the NameNode, the Secondary NameNode can be configured manually to bring up the cluster. However, the Secondary NameNode does not record any real-time changes that happen to the HDFS metadata.

5.10.2 Anatomy of File Read

Figure 5.18 describes the anatomy of File Read.

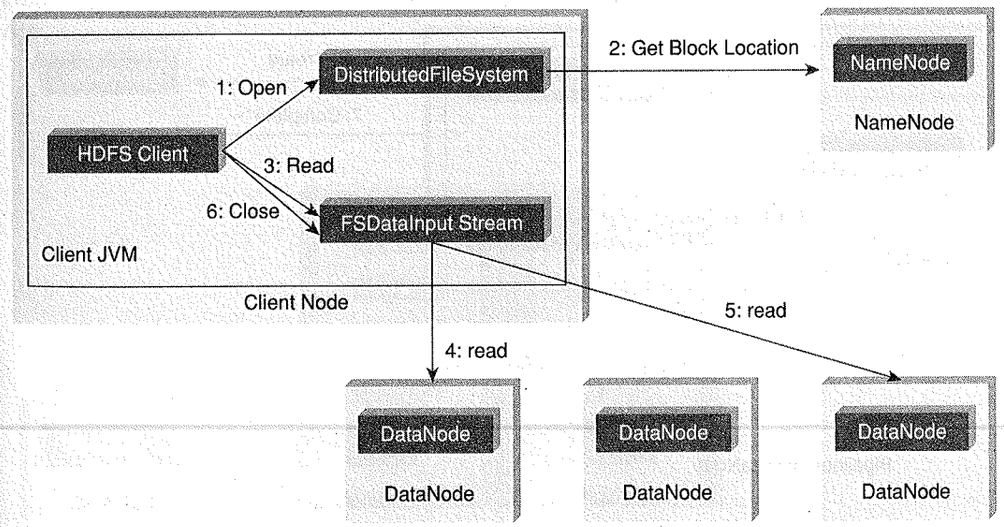


Figure 5.18 File Read.

The steps involved in the File Read are as follows:

1. The client opens the file that it wishes to read from by calling open() on the DistributedFileSystem.
2. DistributedFileSystem communicates with the NameNode to get the location of data blocks. NameNode returns with the addresses of the DataNodes that the data blocks are stored on. Subsequent to this, the DistributedFileSystem returns an FSDataInputStream to client to read from the file.
3. Client then calls read() on the stream DFSInputStream, which has addresses of the DataNodes for the first few blocks of the file, connects to the closest DataNode for the first block in the file.
4. Client calls read() repeatedly to stream the data from the DataNode.
5. When end of the block is reached, DFSInputStream closes the connection with the DataNode. It repeats the steps to find the best DataNode for the next block and subsequent blocks.
6. When the client completes the reading of the file, it calls close() on the FSDataInputStream to close the connection.

Reference: Hadoop, The Definitive Guide, 3rd Edition, O'Reilly Publication.

5.10.3 Anatomy of File Write

Figure 5.19 describes the anatomy of File Write. The steps involved in anatomy of File Write are as follows:

1. The client calls create() on DistributedFileSystem to create a file.
2. An RPC call to the NameNode happens through the DistributedFileSystem to create a new file. The NameNode performs various checks to create a new file (checks whether such a file exists or not). Initially, the NameNode creates a file without associating any data blocks to the file. The DistributedFileSystem returns an FSDataOutputStream to the client to perform write.
3. As the client writes data, data is split into packets by DFSOutputStream, which is then written to an internal queue, called *data queue*. DataStreamer consumes the data queue. The DataStreamer requests

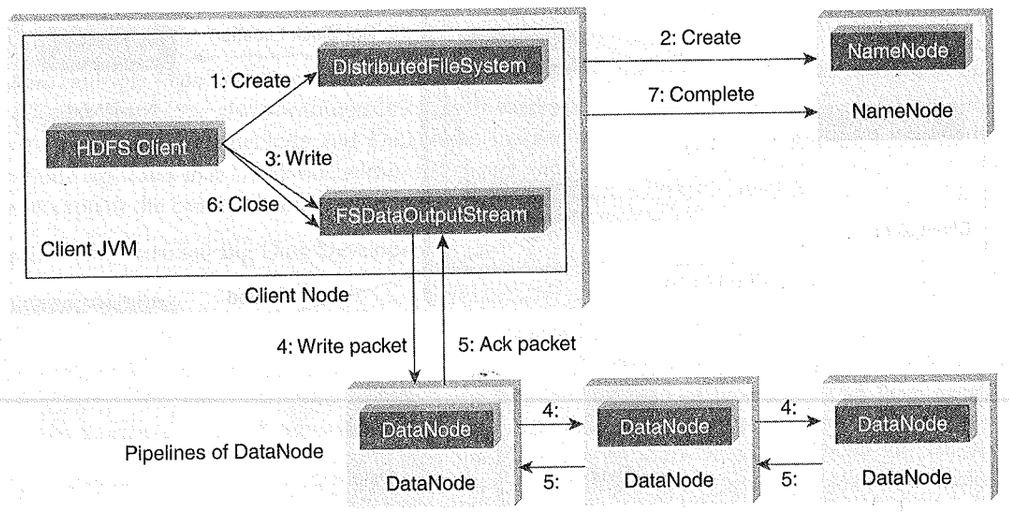


Figure 5.19 File Write.

the NameNode to allocate new blocks by selecting a list of suitable DataNodes to store replicas. This list of DataNodes makes a pipeline. Here, we will go with the default replication factor of three, so there will be three nodes in the pipeline for the first block.

4. DataStreamer streams the packets to the first DataNode in the pipeline. It stores packet and forwards it to the second DataNode in the pipeline. In the same way, the second DataNode stores the packet and forwards it to the third DataNode in the pipeline.
5. In addition to the internal queue, DFSOutputStream also manages an "Ack queue" of packets that are waiting for the acknowledgement by DataNodes. A packet is removed from the "Ack queue" only if it is acknowledged by all the DataNodes in the pipeline.
6. When the client finishes writing the file, it calls close() on the stream.
7. This flushes all the remaining packets to the DataNode pipeline and waits for relevant acknowledgments before communicating with the NameNode to inform the client that the creation of the file is complete.

Reference: Hadoop, The Definitive Guide, 3rd Edition, O'Reilly Publication.

5.10.4 Replica Placement Strategy

5.10.4.1 Hadoop Default Replica Placement Strategy

As per the Hadoop Replica Placement Strategy, first replica is placed on the same node as the client. Then it places second replica on a node that is present on different rack. It places the third replica on the same rack as second, but on a different node in the rack. Once replica locations have been set, a pipeline is built. This strategy provides good reliability. Figure 5.20 describes the typical replica pipeline.

Reference: Hadoop, the Definite Guide, 3rd Edition, O'Reilly Publication.

5.10.5 Working with HDFS Commands

Objective: To get the list of directories and files at the root of HDFS.
Act:
hadoop fs -ls /

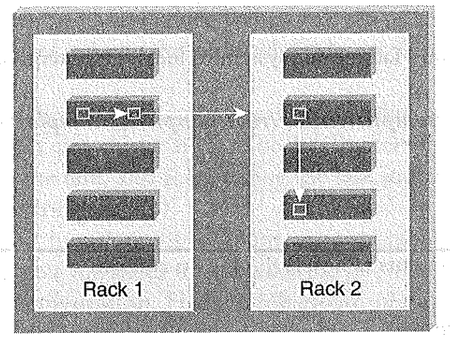


Figure 5.20 Replica Placement Strategy.

Objective: To get the list of complete directories and files of HDFS.
Act:
hadoop fs -ls -R /

Objective: To create a directory (say, sample) in HDFS.
Act:
hadoop fs -mkdir /sample

Objective: To copy a file from local file system to HDFS.
Act:
hadoop fs -put /root/sample/test.txt /sample/test.txt

Objective: To copy a file from HDFS to local file system.
Act:
hadoop fs -get /sample/test.txt /root/sample/testsample.txt

Objective: To copy a file from local file system to HDFS via copyFromLocal command.
Act:
hadoop fs -copyFromLocal /root/sample/test.txt /sample/testsample.txt

Objective: To copy a file from Hadoop file system to local file system via copyToLocal command.
Act:
hadoop fs -copyToLocal /sample/test.txt /root/sample/testsample1.txt

Objective: To display the contents of an HDFS file on console.
Act:
hadoop fs -cat /sample/test.txt

Objective: To copy a file from one directory to another on HDFS.
Act:
hadoop fs -cp /sample/test.txt /sample1

Objective: To remove a directory from HDFS.
Act:
hadoop fs-rm-r /sample1

5.10.6 Special Features of HDFS

1. **Data Replication:** There is absolutely no need for a client application to track all blocks. It directs the client to the nearest replica to ensure high performance.
2. **Data Pipeline:** A client application writes a block to the first DataNode in the pipeline. Then this DataNode takes over and forwards the data to the next node in the pipeline. This process continues for all the data blocks, and subsequently all the replicas are written to the disk.

Reference: Wrox Certified Big Data Developer.

5.11 PROCESSING DATA WITH HADOOP

MapReduce Programming is a software framework. MapReduce Programming helps you to process massive amounts of data in parallel.

In MapReduce Programming, the input dataset is split into independent chunks. **Map tasks** process these independent chunks completely in a parallel manner. The output produced by the map tasks serves as intermediate data and is stored on the local disk of that server. The output of the mappers are automatically shuffled and sorted by the framework. MapReduce Framework sorts the output based on **keys**. This sorted output becomes the input to the **reduce tasks**. Reduce task provides reduced output by combining the output of the various mappers. Job inputs and outputs are stored in a file system. MapReduce framework also takes care of the other tasks such as scheduling, monitoring, re-executing failed tasks, etc.

Hadoop Distributed File System and MapReduce Framework run on the same set of nodes. This configuration allows effective scheduling of tasks on the nodes where data is present **(Data Locality)**. This in turn results in very high throughput.

There are two daemons associated with MapReduce Programming. A single master **JobTracker** per cluster and one slave **TaskTracker** per cluster-node. The JobTracker is responsible for scheduling tasks to the TaskTrackers, monitoring the task, and re-executing the task just in case the TaskTracker fails. The TaskTracker executes the task. Refer Figure 5.21.

The MapReduce functions and input/output locations are implemented via the MapReduce applications. These applications use suitable interfaces to construct the job. The application and the job parameters together are known as **job configuration**. Hadoop **job client** submits job (jar/executable, etc.) to the JobTracker. Then it is the responsibility of JobTracker to schedule tasks to the slaves. In addition to scheduling, it also monitors the task and provides status information to the job-client.

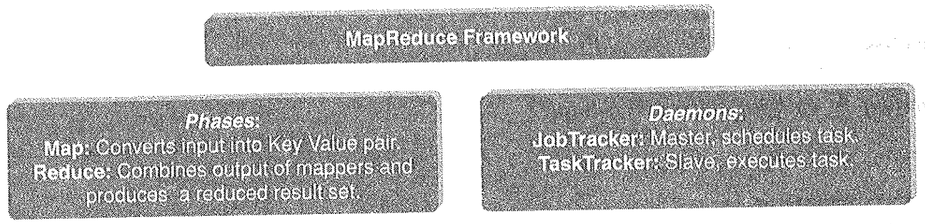


Figure 5.21 MapReduce Programming phases and daemons.

Reference: http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html

5.11.1 MapReduce Daemons

1. **JobTracker:** It provides connectivity between Hadoop and your application. When you submit code to cluster, JobTracker creates the execution plan by deciding which task to assign to which node. It also monitors all the running tasks. When a task fails, it automatically re-schedules the task to a different node after a predefined number of retries. JobTracker is a master daemon responsible for executing overall MapReduce job. There is a single JobTracker per Hadoop cluster.
2. **TaskTracker:** This daemon is responsible for executing individual tasks that is assigned by the JobTracker. There is a single TaskTracker per slave and spawns multiple Java Virtual Machines (JVMs) to handle multiple map or reduce tasks in parallel. TaskTracker continuously sends heartbeat message to JobTracker. When the JobTracker fails to receive a heartbeat from a TaskTracker, the JobTracker assumes that the TaskTracker has failed and resubmits the task to another available node in the cluster. Once the client submits a job to the JobTracker, it partitions and assigns diverse MapReduce tasks for each TaskTracker in the cluster. Figure 5.22 depicts JobTracker and TaskTracker interaction.

Reference: Hadoop in Action, Chuck Lam.

5.11.2 How Does MapReduce Work?

MapReduce divides a data analysis task into two parts – **map** and **reduce**. Figure 5.23 depicts how the MapReduce Programming works. In this example, there are two mappers and one reducer. Each mapper

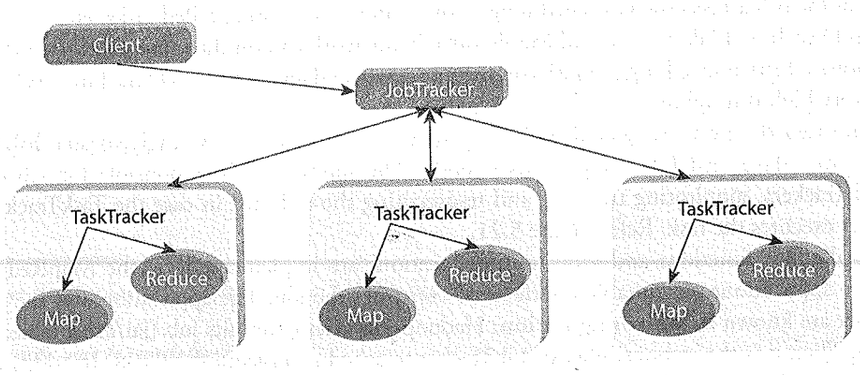


Figure 5.22 JobTracker and TaskTracker interaction.

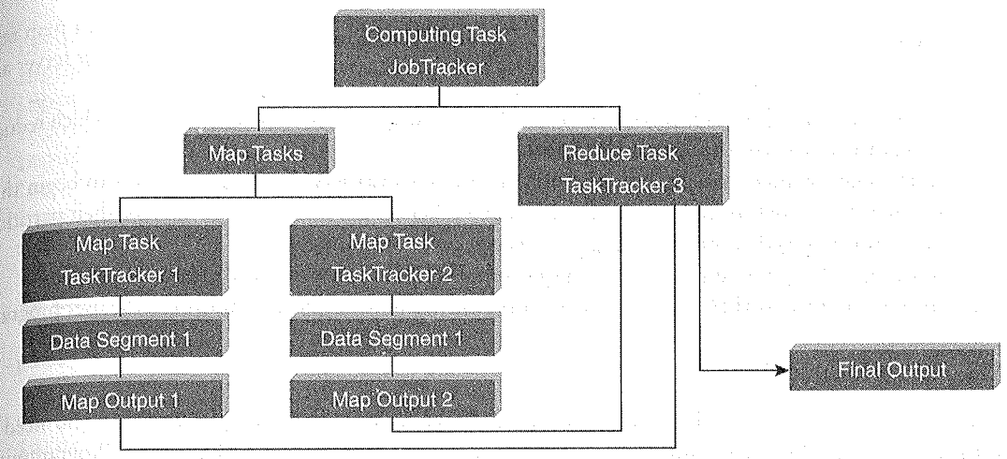


Figure 5.23 MapReduce programming workflow.

works on the partial dataset that is stored on that node and the reducer combines the output from the mappers to produce the reduced result set.

Reference: Wrox Big Data Certification Materials.

Figure 5.24 describes the working model of MapReduce Programming. The following steps describe how MapReduce performs its task.

1. First, the input dataset is split into multiple pieces of data (several small subsets).
2. Next, the framework creates a master and several workers processes and executes the worker processes remotely.

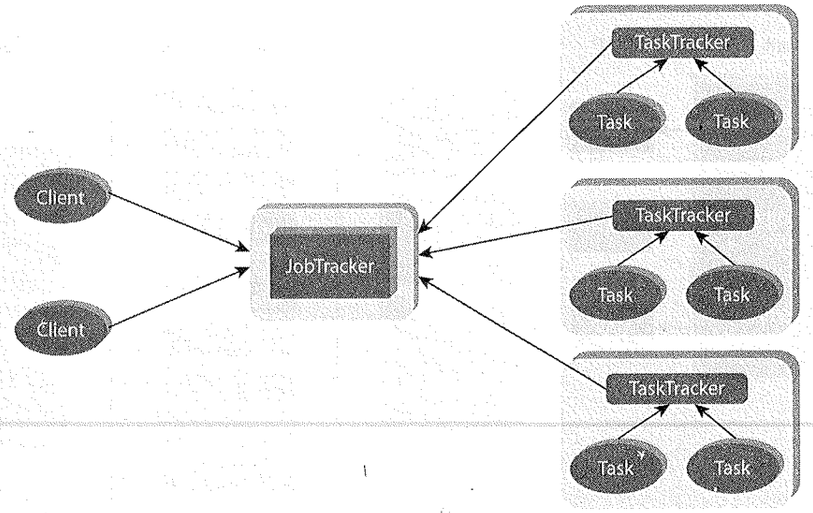


Figure 5.24 MapReduce programming architecture.

3. Several map tasks work simultaneously and read pieces of data that were assigned to each map task. The map worker uses the map function to extract only those data that are present on their server and generates key/value pair for the extracted data.
4. Map worker uses partitioner function to divide the data into regions. Partitioner decides which reducer should get the output of the specified mapper.
5. When the map workers complete their work, the master instructs the reduce workers to begin their work. The reduce workers in turn contact the map workers to get the key/value data for their partition. The data thus received is shuffled and sorted as per keys.
6. Then it calls reduce function for every unique key. This function writes the output to the file.
7. When all the reduce workers complete their work, the master transfers the control to the user program.

5.11.3 MapReduce Example

The famous example for MapReduce Programming is **Word Count.** For example, consider you need to count the occurrences of similar words across 50 files. You can achieve this using MapReduce Programming. Refer Figure 5.25.

Word Count MapReduce Programming using Java

The MapReduce Programming requires three things.

1. **Driver Class:** This class specifies **Job Configuration** details.
2. **Mapper Class:** This class overrides the **Map Function** based on the problem statement.
3. **Reducer Class:** This class overrides the **Reduce Function** based on the problem statement.

Wordcounter.java: Driver Program

```
package com.app;
import java.io.IOException;
```

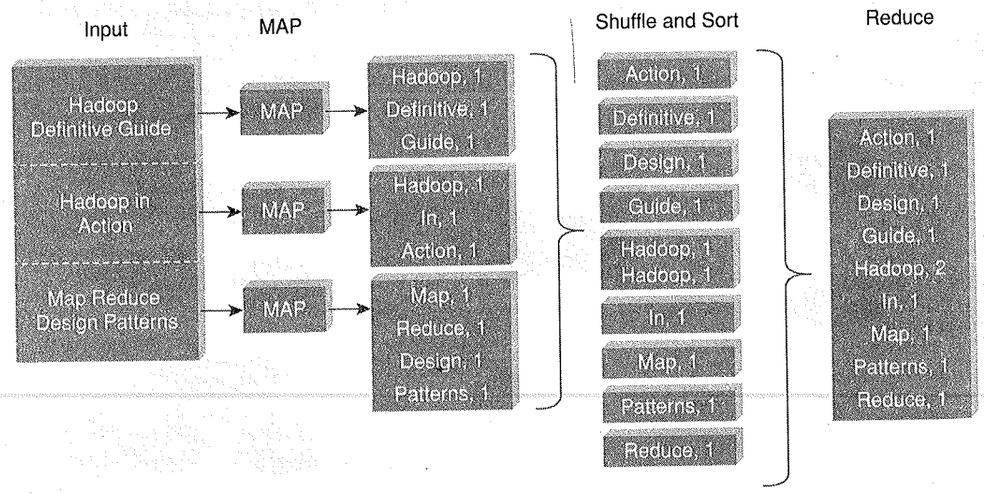


Figure 5.25 Wordcount example.


```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCounter {

public static void main (String [] args) throws IOException,
InterruptedException, ClassNotFoundException {
          Job job = new Job ();
          job.setJobName ("wordcounter");
          job.setJarByClass (WordCounter.class);
          job.setMapperClass (WordCounterMap.class);
          job.setReducerClass (WordCounterRed.class);
          job.setOutputKeyClass (Text.class);
          job.setOutputValueClass (IntWritable.class);

          FileInputFormat.addInputPath (job, new Path ("/sample/word.
txt"));
          FileOutputFormat.setOutputPath (job, new Path ("/sample/
wordcount"));
          System.exit (job.waitForCompletion (true)? 0: 1);

     }
}
```

WordCounterMap.java: Map Class

```
package com.app;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCounterMap extends Mapper<LongWritable, Text, Text,
IntWritable> {
     @Override
```

```
    protected void map (LongWritable key, Text value, Context context)
              throws IOException, InterruptedException {

         String [] words=value.toString ().split (",");
         for (String word: words) {
              context.write (new Text (word), new IntWritable (1));
         }
    }
}
```

WordCountReduce.java: Reduce Class

```
package com.infosys;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCounterRed extends Reducer<Text, IntWritable, Text,
IntWritable>{

     @Override
     protected void reduce(Text word, Iterable<IntWritable> values,
Context context)
               throws IOException, InterruptedException {
          Integer count = 0;
          for(IntWritable val: values){
               count += val.get();
          }
          context.write(word, new IntWritable(count));
     }
}
```

Table 5.2 describes differences between SQL and MapReduce.

Table 5.2 SQL versus MapReduce

	SQL	MapReduce
Access	Interactive and Batch	Batch
Structure	Static	Dynamic
Updates	Read and write many times	Write once, read many times
Integrity	High	Low
Scalability	Nonlinear	Linear

5.12 MANAGING RESOURCES AND APPLICATIONS WITH HADOOP YARN (YET ANOTHER RESOURCE NEGOTIATOR)

Apache Hadoop YARN is a sub-project of Hadoop 2.x. Hadoop 2.x is YARN-based architecture. It is a general processing platform. YARN is not constrained to MapReduce only. You can run multiple applications in Hadoop 2.x in which all applications share a common resource management. Now Hadoop can be used for various types of processing such as Batch, Interactive, Online, Streaming, Graph, and others.

5.12.1 Limitations of Hadoop 1.0 Architecture

In Hadoop 1.0, HDFS and MapReduce are Core Components, while other components are built around the core.

1. Single NameNode is responsible for managing entire namespace for Hadoop Cluster.
2. It has a restricted processing model which is suitable for batch-oriented MapReduce jobs.
3. Hadoop MapReduce is not suitable for interactive analysis.
4. Hadoop 1.0 is not suitable for machine learning algorithms, graphs, and other memory intensive algorithms.
5. **MapReduce** is responsible for **cluster resource management and data processing**.

In this Architecture, **map slots might be "full", while the reduce slots are empty and vice versa**. This causes **resource utilization issues**. This needs to be improved for proper resource utilization.

5.12.2 HDFS Limitation

NameNode saves all its file metadata in main memory. Although the main memory today is not as small and as expensive as it used to be two decades ago, still there is a limit on the number of objects that one can have in the memory on a single NameNode. The NameNode can quickly become overwhelmed with load on the system increasing.

In Hadoop 2.x, this is resolved with the help of **HDFS Federation**.

5.12.3 Hadoop 2: HDFS

HDFS 2 consists of two major components: **(a) namespace**, **(b) blocks storage service**. Namespace service takes care of file-related operations, such as creating files, modifying files, and directories. The block storage service handles data node cluster management, replication.

HDFS 2 Features

1. Horizontal scalability.
2. High availability.

HDFS Federation uses multiple independent NameNodes for horizontal scalability. NameNodes are independent of each other. It means, NameNodes does not need any coordination with each other. The DataNodes are common storage for blocks and shared by all NameNodes. All DataNodes in the cluster registers with each NameNode in the cluster.

High availability of NameNode is obtained with the help of **Passive Standby NameNode**. In Hadoop 2.x, Active–Passive NameNode handles failover automatically. All namespace edits are recorded to a shared NFS storage and there is a single writer at any point of time. Passive NameNode reads edits from shared storage

and keeps updated metadata information. In case of Active NameNode failure, Passive NameNode becomes an Active NameNode automatically. Then it starts writing to the shared storage. Figure 5.26 describes the Active–Passive NameNode interaction.

Reference: http://www.edureka.co/blog/introduction-to-hadoop-2-0-and-advantages-of-hadoop-2-0/

Figure 5.27 depicts Hadoop 1.0 and Hadoop 2.0 architecture.

5.12.4 Hadoop 2 YARN: Taking Hadoop beyond Batch

YARN helps us to store all data in one place. We can interact in multiple ways to get predictable performance and quality of services. This was originally architected by **Yahoo**. Refer Figure 5.28.

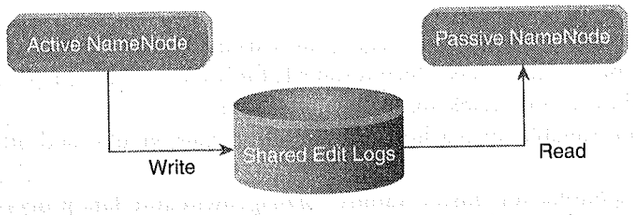


Figure 5.26 Active and Passive NameNode interaction.

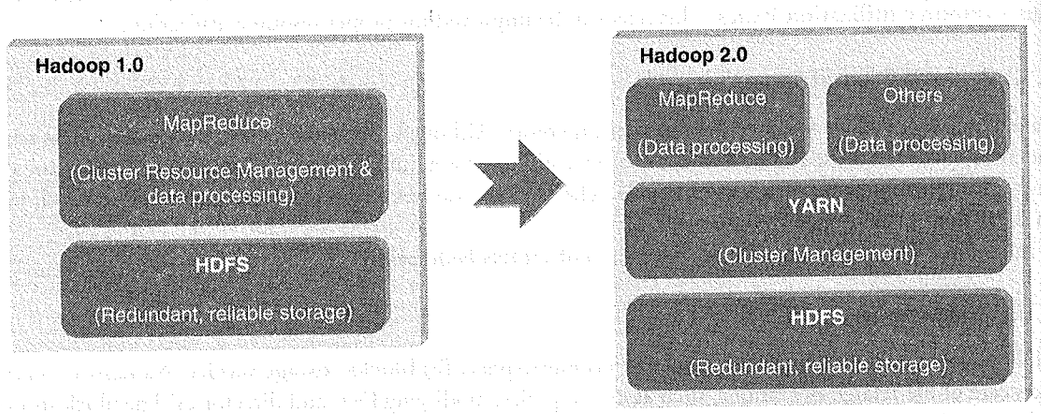


Figure 5.27 Hadoop 1.x versus Hadoop 2.x.

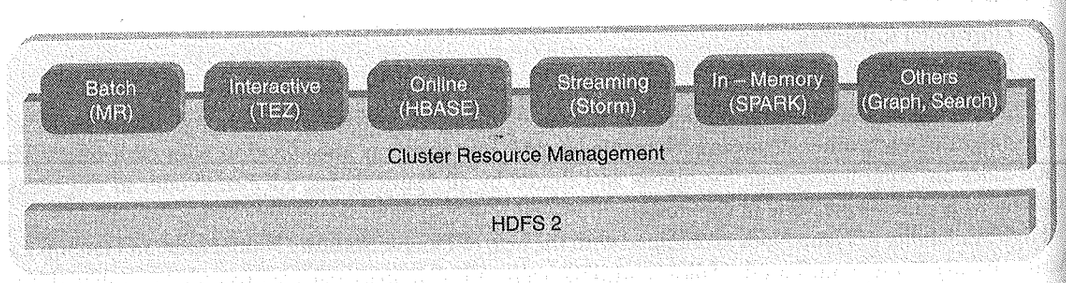


Figure 5.28 Hadoop YARN.

5.12.4.1 Fundamental Idea

The fundamental idea behind this architecture is splitting the JobTracker responsibility of resource management and Job Scheduling/Monitoring into separate daemons. Daemons that are part of YARN Architecture are described below.

1. **A Global ResourceManager:** Its main responsibility is to distribute resources among various applications in the system. It has two main components:
   (a) ***Scheduler:*** The pluggable scheduler of ResourceManager decides allocation of resources to various running applications. The scheduler is just that, a pure scheduler, meaning it does NOT monitor or track the status of the application.
   (b) ***ApplicationManager:*** ApplicationManager does the following:
   - Accepting job submissions.
   - Negotiating resources (container) for executing the application specific ApplicationMaster.
   - Restarting the ApplicationMaster in case of failure.
2. **NodeManager:** This is a per-machine slave daemon. NodeManager responsibility is launching the application containers for application execution. NodeManager monitors the resource usage such as memory, CPU, disk, network, etc. It then reports the usage of resources to the global ResourceManager.
3. **Per-application ApplicationMaster:** This is an application-specific entity. Its responsibility is to negotiate required resources for execution from the ResourceManager. It works along with the NodeManager for executing and monitoring component tasks.

5.12.4.2 Basic Concepts

Application:

1. Application is a job submitted to the framework.
2. Example – **MapReduce Job**.

Container:

1. Basic unit of allocation.
2. Fine-grained resource allocation across multiple resource types (Memory, CPU, disk, network, etc.)
   (a) container_0 = 2GB, 1CPU
   (b) container_1 = 1GB, 6 CPU
3. Replaces the fixed map/reduce slots.

YARN Architecture:

Figure 5.29 depicts YARN architecture. The steps involved in YARN architecture are as follows:

1. A client program submits the application which includes the necessary specifications to launch the application-specific **ApplicationMaster** itself.
2. The ResourceManager launches the ApplicationMaster by assigning some container.
3. The ApplicationMaster, on boot-up, registers with the ResourceManager. This helps the client program to query the ResourceManager directly for the details.
4. During the normal course, ApplicationMaster negotiates appropriate resource containers via the resource-request protocol.

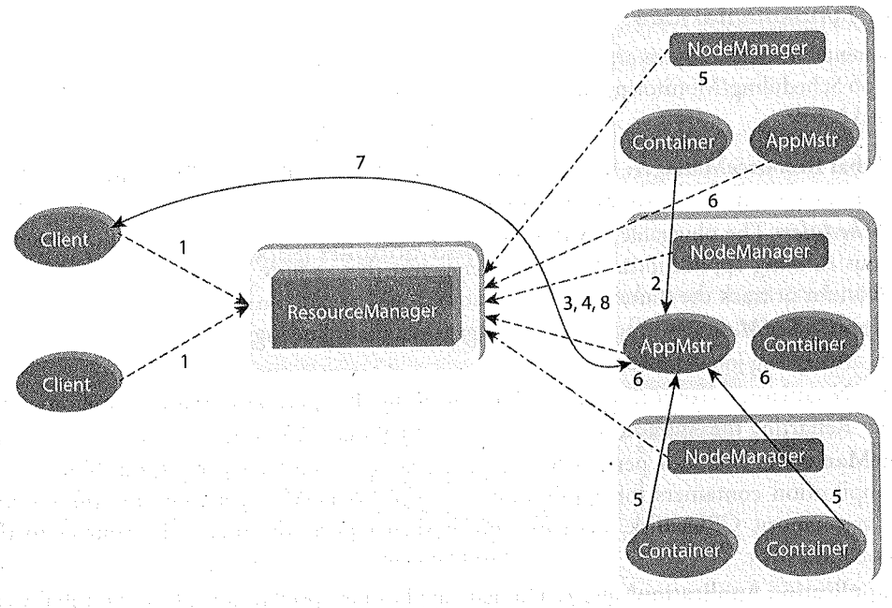


Figure 5.29 YARN architecture.

5. On successful container allocations, the ApplicationMaster launches the container by providing the container launch specification to the NodeManager.
6. The NodeManager executes the application code and provides necessary information such as progress, status, etc. to it's ApplicationMaster via an application-specific protocol.
7. During the application execution, the client that submitted the job directly communicates with the ApplicationMaster to get status, progress updates, etc. via an application-specific protocol.
8. Once the application has been processed completely, ApplicationMaster deregisters with the ResourceManager and shuts down, allowing its own container to be repurposed.

Reference: http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/

5.13 INTERACTING WITH HADOOP ECOSYSTEM

Hadoop ecosystem was introduced in Chapter 4. Here we will look at it in more detail.

5.13.1 Pig

Pig is a data flow system for Hadoop. It uses Pig Latin to specify data flow. Pig is an alternative to MapReduce Programming. It abstracts some details and allows you to focus on data processing. It consists of two components.

1. **Pig Latin:** The data processing language.
2. **Compiler:** To translate Pig Latin to MapReduce Programming.

Figure 5.30 depicts the Pig in the Hadoop ecosystem.

5.13.2 Hive

Hive is a Data Warehousing Layer on top of Hadoop. Analysis and queries can be done using an SQL-like language. Hive can be used to do ad-hoc queries, summarization, and data analysis. Figure 5.31 depicts Hive in the Hadoop ecosystem.

5.13.3 Sqoop

Sqoop is a tool which helps to transfer data between Hadoop and Relational Databases. With the help of Sqoop, you can import data from RDBMS to HDFS and vice-versa. Figure 5.32 depicts the Sqoop in Hadoop ecosystem.

5.13.4 HBase

HBase is a NoSQL database for Hadoop. HBase is column-oriented NoSQL database. HBase is used to store **billions of rows and millions of columns.** HBase provides random read/write operation. It also supports record level updates which is not possible using HDFS. HBase sits on top of HDFS. Figure 5.33 depicts the HBase in Hadoop ecosystem.

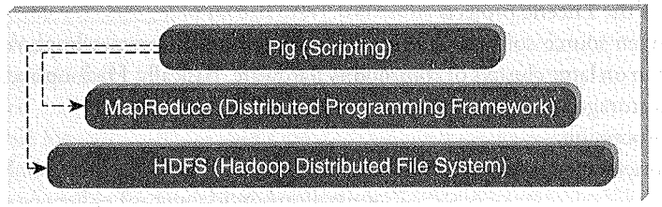


Figure 5.30 Pig in the Hadoop ecosystem.

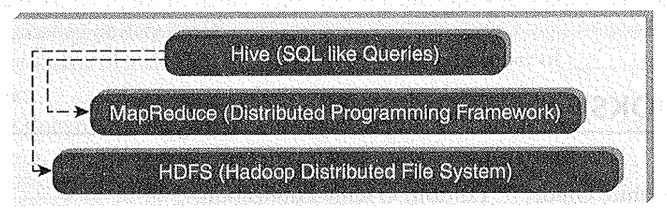


Figure 5.31 Hive in the Hadoop ecosystem.

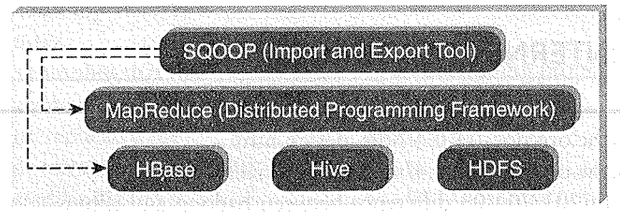


Figure 5.32 Sqoop in the Hadoop ecosystem.

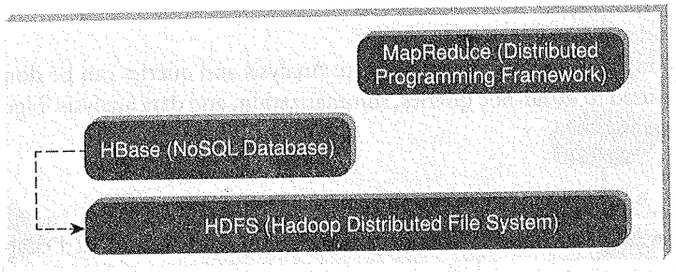


Figure 5.33 HBase in the Hadoop Ecosystem.

REMIND ME

- The key consideration (the rationale behind the huge popularity of Hadoop) is: ***Its capability to handle massive amounts of data, different categories of data – fairly quickly.***
- Hadoop was created by Doug Cutting, the creator of Apache Lucene (a commonly used text search library). Hadoop is a part of the Apache Nutch (Yahoo) project (an open-source web search engine) and also a part of the Lucene project.
- Hadoop is an open-source software framework. It stores and processes huge volumes of data in a distributed fashion on large clusters of commodity hardware. Basically, Hadoop accomplishes two tasks:
  - Massive data storage.
  - Faster data processing.
- The core components of Hadoop are:
  - HDFS
  - MapReduce
- Apache Hadoop YARN is a sub-project of Hadoop 2.x. Hadoop 2.x is YARN-based architecture. It provides general processing platform which is not constrained to MapReduce only.

POINT ME (BOOKS)

- Hadoop, the Definite Guide, 3rd Edition, O'reilly Publication.
- Hadoop in Practice, Alex Holmes.
- Hadoop in Action, Chuck Lam.

CONNECT ME (INTERNET RESOURCES)

- http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html
- http://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html
- http://oraclesys.com/2013/04/03/difference-between-hadoop-and-rdbms/

- http://hortonworks.com/blog/apache-hadoop-yarn-background-and-an-overview/
- http://www.tomsitpro.com/articles/hadoop-2-vs-1,2-718.html
- http://www.wikidifference.com/difference-between-hadoop-and-rdbms/
- http://www.edureka.co/blog/introduction-to-hadoop-2-0-and-advantages-of-hadoop-2-0/

TEST ME

A. Fill Me

1. Hadoop is _____ based flat structure.
2. RDBMS is best choice when _____ is the main concern.
3. Hadoop supports _____, _____ and _____ data formats.
4. RDBMS supports _____ data formats.
5. In Hadoop, data is processed in _____.
6. HDFS can be deployed on _____.
7. NameNode uses _____ to store file system namespace.
8. NameNode uses _____ to record every transaction.
9. Secondary NameNode is a _____ daemon.
10. DataNode is responsible for _____ file operation.
11. Hadoop 2.x is based on _____ architecture.
12. YARN is responsible for _____.
13. Global ResourceManager distributes _____ among applications.
14. NodeManager is responsible for launching Application _____.
15. Application is a _____ submitted to framework.
16. _____ is an open-source framework managed by Apache Software Foundations.
17. The emphasis of HDFS is on _____ throughput of data access rather than _____ latency of data access.
18. An HDFS cluster consists of a single _____ and a number of _____.
19. Complete the series:
    Bits → Bytes → Kilobytes → Megabytes → Gigabytes → _____ → _____ → _____ → _____ → Yottabytes
20. HDFS has a _____/_____ architecture.
21. HDFS is built using the _____ language.
22. The _____ maintains the file system Namespace.
23. The number of copies of a file is called the _____ of that file.
24. The NameNode periodically receives a _____ and a _____ from each of the DataNodes in the cluster.
25. Receipt of a Heartbeat implies that the _____ is functioning properly.
26. A _____ contains a list of all blocks on a DataNode.
27. The blocks of a file are replicated for _____ tolerance.
28. When the NameNode starts up, it reads the _____ and _____ from disk.
29. A typical block size used by HDFS is _____.
30. _____ are responsible for serving read and write requests from the file system's clients.

31. _____ perform block creation, deletion and replication upon instruction from the _____.
32. _____ was the first to publicize MapReduce – a system they had used to scale their data processing needs.
33. _____ developed an open-source version of MapReduce system called _____.
34. Hadoop is an open-source framework for writing and running _____ applications that process large amounts of data.
35. The key distinctions of Hadoop are _____, _____ and _____.
36. Hadoop runs on large clusters of _____.
37. Hadoop scales _____ to handle larger data by adding more _____ to the cluster.
38. Hadoop focusses on moving _____ to _____.
39. The move-code-to-data philosophy makes sense for _____ intensive processing.
40. Hadoop is designed to be a scale _____ architecture operating on cluster of commodity PC machines.
41. Hadoop uses _____ as its basic data unit, which is flexible enough to work with less-structured data types.
42. Hadoop is best used as a _____ once and _____ many times type of data store.
43. Under SQL we have _____ statements; under MapReduce we have _____ and _____.
44. Under the MapReduce Model, data processing primitives are called _____ and _____.
45. The Mapper is meant to _____ and _____ the input into something that the reducer can _____ over.
46. _____ and _____ are common design patterns that go along with mapping and reducing.
47. _____ is the official development and production platform for Hadoop.
48. _____ started out as a sub-project of _____, which in turn was a sub-project of _____.
49. _____ is a single point of failure of Hadoop cluster.
50. _____ is the bookkeeper of HDFS.
51. _____ keeps track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed file system.
52. _____ communicates with the NameNode to take snapshots of the HDFS metadata at intervals defined by the cluster configuration.
53. There is only one _____ daemon per Hadoop cluster.
54. There is a single _____ per slave node.

Answers:

1. Node
2. Consistency
3. Structured, semi-structured and unstructured
4. Structured
5. Parallel
6. Low cost hardware
7. FsImage
8. EditLog

9. Helper or House Keeping
10. Read/Write
11. YARN
12. Cluster Management
13. Resources
14. Containers
15. Job
16. Hadoop
17. High, Low
18. NameNode, DataNodes
19. Terabytes, Petabytes, Exabytes, Zettabytes
20. Master/slave
21. Java
22. NameNode
23. Replication factor
24. Heartbeat, Blockreport
25. DataNode
26. Blockreport
27. Fault
28. FsImage, EditLog
29. 64MB
30. DataNodes
31. DataNodes, NameNode
32. Google
33. Doug Cutting, Hadoop
34. Distributed
35. Accessible, Robust, and Scalable
36. Commodity machines
37. Linearly, nodes
38. Code, Data
39. Data
40. Out
41. Key/value
42. Write, read
43. Query, Scripts, and Code
44. Mappers, Reducers
45. Filter and transform, aggregate
46. Partitioning and Shuffling
47. Linux
48. Hadoop, Nutch, Apache Lucene
49. NameNode
50. NameNode
51. NameNode
52. Secondary NameNode
53. JobTracker
54. TaskTracker

B. Match Me

1.

Column A	Column B
HDFS	DataNode
MapReduce Programming	NameNode
Master node	Processing Data
Slave node	Google File System and MapReduce
Hadoop Implementation	Storage

Answer:

Column A	Column B
HDFS	Storage
MapReduce Programming	Processing Data
Master node	NameNode
Slave node	DataNode
Hadoop Implementation	Google File System and MapReduce

2.

Column A	Column B
JobTracker	Executes Task
MapReduce	Schedules Task
TaskTracker	Programming Model
Job Configuration	Converts input into Key Value pair
Map	Job Parameters

Answer:

Column A	Column B
JobTracker	Schedules Task
MapReduce	Programming Model
TaskTracker	Executes Task
Job Configuration	Job Parameters
Map	Converts input into Key Value pair

3.

Column A	ColumnB
NameNode	Handles processing on master
JobTracker	Handles storage on slave
DataNode	Handles storage on master
TaskTracker	Handles processing on slave

Answer:

Column A	Column B
NameNode	Handles storage on master
JobTracker	Handles processing on master
DataNode	Handles storage on slave
TaskTracker	Handles processing on slave

C. True or False

1. For using Hadoop to process your data, the data has to be moved/ingested into HDFS.
2. Sqoop is used to query HDFS data.

3. Oozie is to import/export data from RDBMS.
4. "hadoop fs -ls /" will show the contents for the HDFS root directory.
5. Master node in Hadoop can be low on disk space but needs to have good amount of RAM.
6. In Production, NameNode preferably runs on Red Hat OS.
7. Hadoop configurations are stored in CSV format.

Answers:

1. True
2. False
3. False
4. True
5. True
6. True
7. False

D. Pick the Right Choice

1. Which of the two are components of Hadoop?
   (a) HDFS
   (b) MapReduce
   (c) Secondary NameNode
   (d) Shuffler
   (e) Sqoop
2. How many blocks will be created for a file that is 300 MB? The default block size is 64 MB and the replication factor is 3.
   (a) 30
   (b) 5
   (c) 15
   (d) 100
3. Pig is a
   (a) Data flow language
   (b) Scheduling engine
   (c) Import export tool
   (d) Shuffler
4. What does JobTracker do?
   (a) Stores blocks of data
   (b) Coordinates and schedules the job
   (c) Stores metadata
   (d) Acts as a mini reducer
5. Which ecosystem project is ideal for use when we have multiple MapReduce and Pig programs to run in a sequence?
   (a) Oozie
   (b) Pig
   (c) Hive
   (d) Sqoop
6. Which file is used for updating MapReduce settings?
   (a) core-site
   (b) hdfs-site
   (c) mapred-site
   (d) hadoop-env.sh

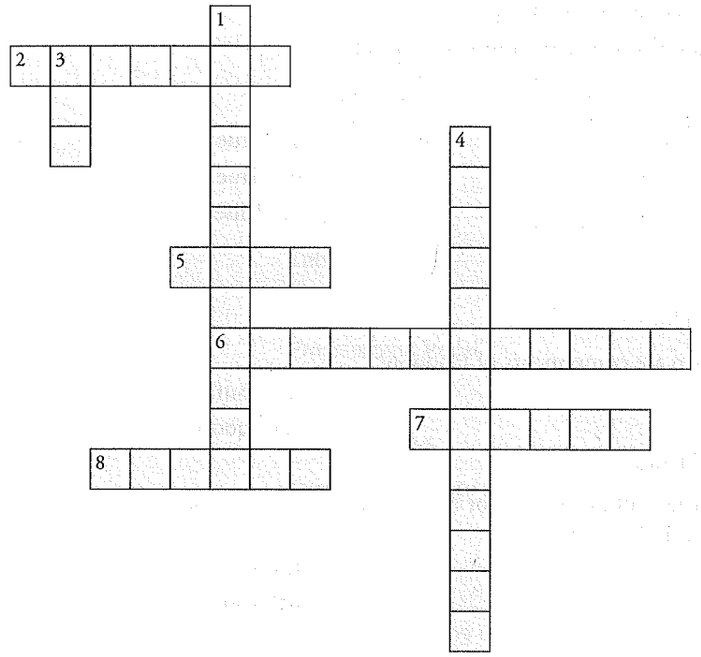
Answers:

1. (a) and (b)
2. (c)
3. (a)
4. (b)
5. (a)
6. (c)

E. Crossword

Puzzle on Big Data and Hadoop

Complete the crossword below



Across

2. One _____ Gigabytes are there in one Exabyte.
5. _____ is Splunk's new product to search, access and report on Hadoop data sets.
6. _____ gave Hadoop its name
7. _____ open-source software was developed from Google's MapReduce concept.
8. The MapReduce programming model widely used in analytics was developed at _____.

Down

1. _____ created the popular Hadoop software framework for storage and processing of large datasets.
3. _____ traditional IT Company is the largest Big Data vendor in the world.
4. According to a study by IBM, approximately _____ amount of data existed in the digital universe in 2012.

Answer:

Across

2. Billion
5. Hunk
6. Toy Elephant
7. Hadoop
8. Google

Down

1. Doug cutting
3. IBM
4. 2.7 Zettabytes

CHALLENGE ME

There are questions on topics that are not covered in the chapter. We will need you to read up on your own.

1. **What are the four modules that make up the Apache Hadoop framework?**

   Answer:

   - Hadoop Common, which contains the common utilities and libraries necessary for Hadoop's other modules.
   - Hadoop YARN, the framework's platform for resource management.
   - Hadoop Distributed File System, or HDFS, which stores information on commodity machines.
   - Hadoop MapReduce, a programming model used to process large-scale sets of data.

2. **Which modes can Hadoop be run in? List a few features for each mode.**

   Answer:

   - Standalone, or local mode, which is one of the least commonly used environments. When it is used, it's usually only for running MapReduce programs. Standalone mode lacks a distributed file system and uses a local file system instead.
   - Pseudo-distributed mode, which runs all daemons on a single machine. It is most commonly used in QA and development environments.
   - Fully distributed mode, which is most commonly used in production environments. Unlike pseudo-distributed mode, fully distributed mode runs all daemons on a cluster of machines rather than a single one.

3. **Where are Hadoop's configuration files located?**

   Answer: Hadoop's configuration files can be found inside the conf sub-directory.

4. **List Hadoop's three configuration files.**

   Answer:

   - hdfs-site.xml
   - core-site.xml
   - mapred-site.xml

5. **How many NameNodes can run on a single Hadoop cluster?**

   Answer: Only one NameNode process can run on a single Hadoop cluster. The file system will go offline if this NameNode goes down.

6. **What is a DataNode?**

   Answer: Unlike NameNode, a DataNode actually stores data within the Hadoop distributed file system. DataNodes run on their own Java virtual machine process.

7. **How many data nodes can run on a single Hadoop cluster?**

   Answer: Hadoop slave nodes contain only one data node process each.

8. **What is JobTracker in Hadoop?**

   Answer: JobTracker is used to submit and track jobs in MapReduce.

9. **How many JobTracker processes can run on a single Hadoop cluster?**

   Answer: There can only be one JobTracker process running on a single Hadoop cluster. JobTracker processes run on their own Java virtual machine process. If JobTracker goes down, all currently active jobs stop.

10. What is the difference between replication and sharding?

Answer: Replication essentially takes the same data and copies it over several machines/nodes (the number of copies it makes, depends on the defined replication factor).

Sharding takes different data and places it on different machines. It is particularly valuable for performance as it can help with read and write operations. Replication is for fault tolerance.

11. What is polyglot persistence?

Answer: The official definition of polyglot is "a person who has the ability to speak, read, and write several languages". Now consider an organization that has grown over 35 years. It has a lot of applications which write to a number of data sources (RDBMSs, Flat files, .xls, csv files, etc.). The organization also has several data marts, content management server, etc. This is a typical polyglot situation as an analytics application may require the data to be read from all of these different types of data sources.

Consider a scenario: You are one of the sponsors of an online retail firm.

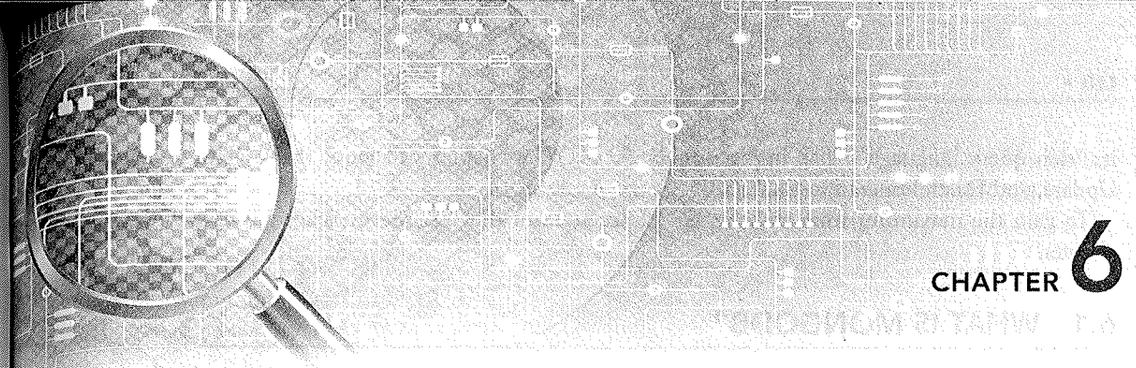
You have a few questions which you need answered:

- Who are the customers who have purchased a product X in the last 12 months?
- Do you have comments left by these customers on social network site?
- Are there repeat customers on the company's website?
- Have they recommended your product to their friends, colleagues, and relatives?
- Did they go to check the product elsewhere?

This calls for data to be collected from varied disparate data sources (relational and non-relational) and analyzed. The above is a typical case of polyglot persistence.

12. What is BigTable?

Answer: It is a compressed, proprietary data storage system built on Google File System. It is not distributed outside of Google, although it underlies the Google Datastore.



CHAPTER 6

Introduction to MongoDB

BRIEF CONTENTS

- What's in Store?
- What is MongoDB?
- Why MongoDB?
  - Using JSON
  - Creating or Generating a Unique Key
  - Support for Dynamic Queries
  - Storing Binary Data
  - Replication
  - Sharding
  - Updating Information In-Place
- Terms used in RDBMS and MongoDB
- Data Types in MongoDB
- MongoDB Query Language: CRUD (Create, Read, Update, and Delete)
  - Insert(), Update(), Save(), Remove(), find()
  - Null Values
  - Count, Limit, Sort, and Skip
  - Arrays
  - Aggregate Function
  - MapReduce Function
  - Java Script Programming
  - Cursors in MongoDB
  - Indexes
  - MongoImport
  - MongoExport
  - Automatic Generation of Unique Numbers for the "_id" Field

"You can have data without information, but you cannot have information without data."

Daniel Keys Moran, computer programmer and science fiction author

WHAT'S IN STORE?

The relational database model has prevailed for decades. Of late a new kind of database is gaining ground in the enterprise called NoSQL (Not only SQL). The focus of this chapter will be on exploring a NoSQL database called "MongoDB". We bring to you the features of MongoDB such as "Auto Sharding", "Replication",

db.books.save({ _id:5,Category:"Web Mining", Bookname:" Learning R ", Author:" Richard Cotton",qty:5, price:850,rol:10,pages:120});

Step 2: Confirm the presence of the above documents in the "books" collection.

Step 3: Write map and reduce functions to split the books into the following two categories:
(a) Big books
(b) Small books

Books which have more than 300 pages should be in the big book category. Books which have less than 300 pages should be in the small book category.

Step 4: Count the number of books in each category.

Step 5: Store the output as follows as documents in a new collection, called, "Book_Result".

Book Category	Count of the Books
(a) Big books	2
(b) Small books	3

ASSIGNMENT 2

Objective: To practice import, export, and aggregation in MongoDB.

Step 1: Pick any public dataset from the site www.kdnuggets.com. Convert it into CSV format. Make sure that you have at least two numeric columns.

Step 2: Use MongoImport to import data from the CSV format file into MongoDB collection, "MongoDBHandsOn" in test database.

Step 3: Identify a grouping column.

Step 4: Compute the sum of the values in the first numeric column.

Step 5: Compute the average of the values in the second numeric column.

ASSIGNMENT 3

Objective: To copy the JSON documents from one MongoDB collection to another MongoDB collection.

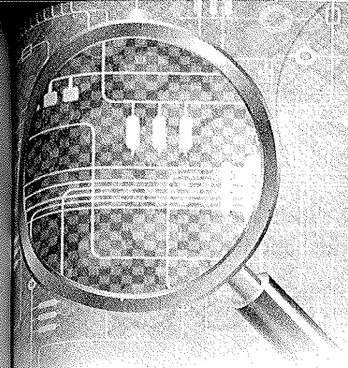
ASSIGNMENT 4

Objective: Write the insert method to store the following document in MongoDB.

Name: "Stephen More"

Address:
{ "City" : "Bangalore",
"Street" : "Electronics City",
"Affiliation" : "XYZ Ltd"
}

Hobbies: Chess, Lawn Tennis, Base ball



CHAPTER 7

Introduction to Cassandra

BRIEF CONTENTS

- What's in Store?
- Apache Cassandra – An Introduction
- Features of Cassandra
  - Peer-to-Peer Network
  - Gossip and Failure Detection
  - Partitioner
  - Replication Factor
  - Anti-Entropy and Read Repair
  - Writes in Cassandra
  - Hinted Handoffs
  - Tunable Consistency: Read Consistency and Write Consistency
- CQL Data Types
- CQLSH
- Keyspaces
- CRUD Operations
  - Insert
  - Update
  - Delete
  - Select
- Collections
  - Set Collection
  - List Collection
  - Map Collection
- Using a Counter
- Time To Live (TTL)
- Alter Commands
  - Alter Table to Change the Data Type of a Column
  - Alter Table to Delete a Column
  - Drop a Table
  - Drop a Database
- Import and Export
  - Export to CSV
  - Import from CSV
  - Import from STDIN
  - Export to STDOUT
- Querying System Tables
- Practice Examples

"Data is a precious thing and will last longer than the systems themselves."

– Tim Berners-Lee, inventor of the World Wide Web.

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.

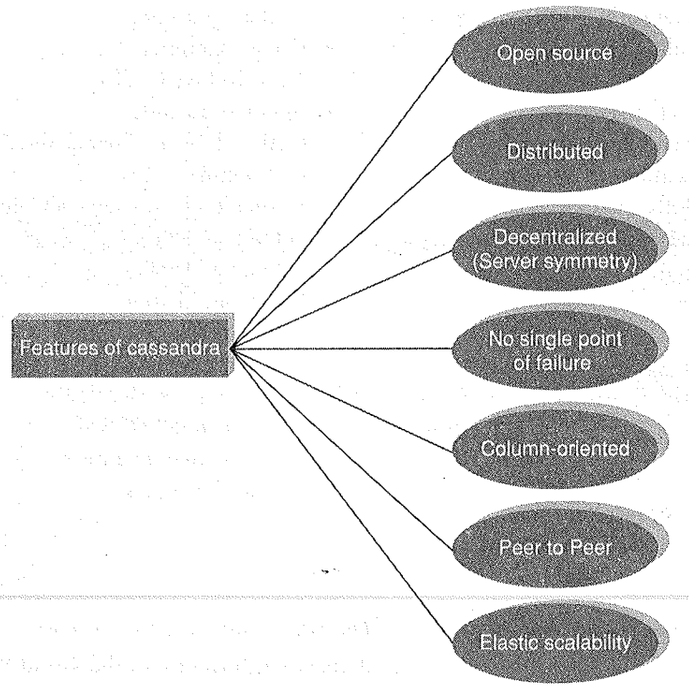


Figure 7.1 Features of Cassandra.

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

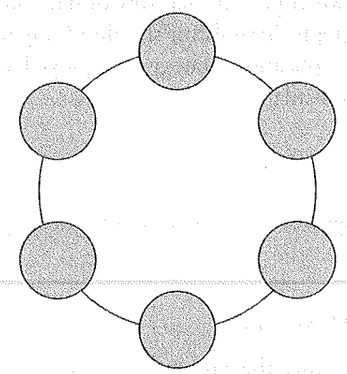


Figure 7.2 Sample Cassandra cluster.

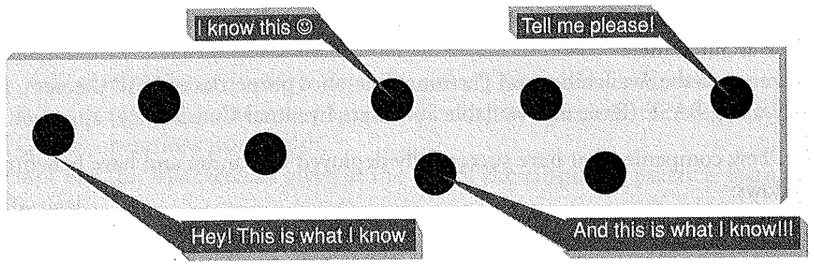


Figure 7.3 Gossip protocol.

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

can connect to any node in the cluster to read data. How many nodes will be read before responding to the client is based on the consistency level specified by the client. If the client-specified consistency is not met, the read operation blocks. There is a possibility that few of the nodes may respond with an out-of-date value. In such a case, Cassandra will initiate a read repair operation to bring the replicas with stale values up to date.

For repairing unread data, Cassandra uses an anti-entropy version of the gossip protocol. Anti-entropy implies comparing all the replicas of each piece of data and updating each replica to the newest version. The read repair operation is performed either before or after returning the value to the client as per the specified consistency level.

7.2.6 Writes in Cassandra

Let us look at behind the scene activities. Here is a client that initiates a write request. Where does his write get written to? It is first written to the commit log. A write is taken as successful only if it is written to the commit log. The next step is to push the write to a memory resident data structure called Memtable. A threshold value is defined in the Memtable. When the number of objects stored in the Memtable reaches a threshold, the contents of Memtable are flushed to the disk in a file called SSTable (Sorted String Table). Flushing is a non-blocking operation. It is possible to have multiple Memtables for a single column family. One out of them is current and the rest are waiting to be flushed.

7.2.7 Hinted Handoffs

The first question that arises is: Why Cassandra is all for availability? It works on the philosophy that it will always be available for writes.

Assume that we have a cluster of three nodes – Node A, Node B, and Node C. Node C is down for some reason. Refer Figure 7.4. We are maintaining a replication factor of 2 which implies that two copies of each row will be stored on two different nodes. The client makes a write request to Node A. Node A is the coordinator and serves as a proxy between the client and the nodes on which the replica is to be placed. The client

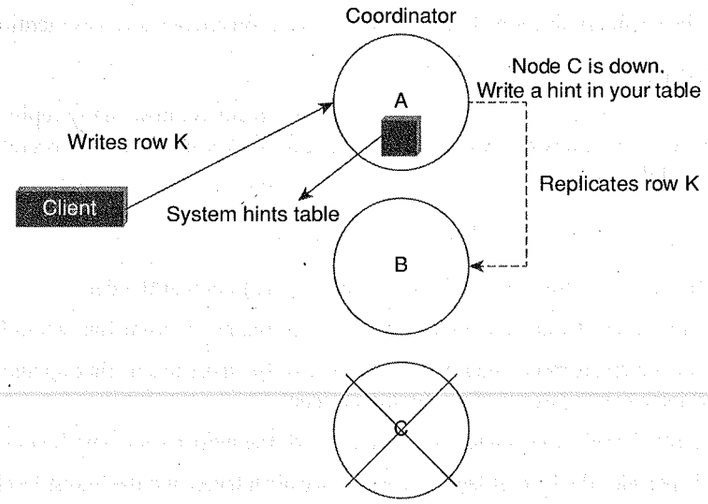


Figure 7.4 Depiction of hinted handoffs.

writes Row K to Node A. Node A then writes Row K to Node B and stores a hint for Node C. The hint will have the following information:

1. Location of the node on which the replica is to be placed.
2. Version metadata.
3. The actual data.

When Node C recovers and is back to the functional self, Node A reacts to the hint by forwarding the data to Node C.

7.2.8 Tunable Consistency

One of the features of Cassandra that has made it immensely popular is its ability to utilize tunable consistency. The database systems can go for either strong consistency or eventual consistency. Cassandra can cash in on either flavor of consistency depending on the requirements. In a distributed system, we work with several servers in the system. Few of these servers are in one data center and others in other data centers. Let us take a look at what it means by strong consistency and eventual consistency.

1. **Strong consistency:** If we work with strong consistency, it implies that each update propagates to all locations where that piece of data resides. Let us assume a single data center setup. Strong consistency will ensure that all of the servers that should have a copy of the data, will have it, before the client is acknowledged with a success. If we are wondering whether it will impact performance, yes it will. It will cost a few extra milliseconds to write to all servers.
2. **Eventual consistency:** If we work with eventual consistency, it implies that the client is acknowledged with a success as soon as a part of the cluster acknowledges the write. When should one go for eventual consistency? The choice is fairly obvious... when application performance matters the most. Example: A single server acknowledges the write and then begins propagating the data to other servers.

7.2.8.1 Read Consistency

Let us understand what the read consistency level means. It means how many replicas must respond before sending out the result to the client application. There are several read consistency levels as mentioned in Table 7.1.

7.2.8.2 Write Consistency

Let us understand what the write consistency level means. It means on how many replicas write must succeed before sending out an acknowledgement to the client application. There are several write consistency levels as mentioned in Table 7.2.

Table 7.1 Read consistency levels in Cassandra

ONE	Returns a response from the closest node (replica) holding the data.
QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data.
LOCAL_ QUORUM	Returns a result from a quorum of servers with the most recent timestamp for the data in the same data center as the coordinator node.
EACH_QUORUM	Returns a result from a quorum of servers with the most recent timestamp in all data centers.
ALL	This provides the highest level of consistency of all levels and the lowest level of availability of all levels. It responds to a read request from a client after all the replica nodes have responded.

Table 7.2 Write consistency levels in Cassandra

ALL	This is the highest level of consistency of all levels as it necessitates that a write must be written to the commit log and Memtable on all replica nodes in the cluster.
EACH_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in *all* data centers.
QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes.
LOCAL_QUORUM	A write must be written to the commit log and Memtable on a quorum of replica nodes in the same data center as the coordinator node. This is to avoid latency of inter-data center communication.
ONE	A write must be written to the commit log and Memtable of at least one replica node.
TWO	A write must be written to the commit log and Memtable of at least two replica nodes.
THREE	A write must be written to the commit log and Memtable of at least three replica nodes.
LOCAL_ONE	A write must be sent to, and successfully acknowledged by, at least one replica node in the local data center.

7.3 CQL DATA TYPES

Refer Table 7.3 for built-in data types for columns in CQL.

Table 7.3 Built-in data types in Cassandra

Int	32 bit signed integer
Bigint	64 bit signed long
Double	64-bit IEEE-754 floating point
Float	32-bit IEEE-754 floating point
Boolean	True or false
Blob	Arbitrary bytes, expressed in hexadecimal
Counter	Distributed counter value
Decimal	Variable – precision integer
List	A collection of one or more ordered elements
Map	A JSON style array of elements
Set	A collection of one or more elements
Timestamp	Date plus time
Varchar	UTF 8 encoded string
Varint	Arbitrary-precision integers
Text	UTF 8 encoded string

7.4 CQLSH

7.4.1 Logging into cqlsh

The below screenshot depicts the cqlsh command prompt after logging in, using cqlsh succeeds.

```
C:\windows\system32\cmd.exe - Python cqlsh

d:\apache-cassandra-2.0.0\apache-cassandra-2.0.0\apache-cassandra-2.0.0\bin>Python cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.0.0 | Cassandra 2.0.0 | CQL spec 3.1.0 | Thrift protocol 19.37.0]
Use HELP for help.
cqlsh>
```

The upcoming sections have been designed as follows:

Objective:	What is it that we are trying to achieve here?
Input (optional):	What is the input that has been given to us to act upon?
Act:	The actual statement /command to accomplish the task at hand.
Outcome:	The result/output as a consequence of executing the statement.

Objective: To get help with CQL.

Act:

```
Help
```

Outcome:

```
C:\windows\system32\cmd.exe - Python cqlsh

d:\apache-cassandra-2.0.0\apache-cassandra-2.0.0\apache-cassandra-2.0.0\bin>Python cqlsh
Connected to Test Cluster at localhost:9160.
[cqlsh 4.0.0 | Cassandra 2.0.0 | CQL spec 3.1.0 | Thrift protocol 19.37.0]
Use HELP for help.
cqlsh> help

Documented shell commands:
==========================
CAPTURE      COPY  DESCRIBE  EXPAND  SHOW    TRACING
CONSISTENCY  DESC  EXIT      HELP    SOURCE

CQL help topics:
================
ALTER                          CREATE_TABLE_OPTIONS  REVOKE
ALTER_ADD                      CREATE_TABLE_TYPES    SELECT
ALTER_ALTER                    CREATE_USER           SELECT_COLUMNFAMILY
ALTER_DROP                     DELETE                SELECT_EXPR
ALTER_RENAME                   DELETE_COLUMNS        SELECT_LIMIT
ALTER_USER                     DELETE_USING          SELECT_TABLE
ALTER_WITH                     DELETE_WHERE          SELECT_WHERE
APPLY                          DROP                  TEXT_OUTPUT
ASCII_OUTPUT                   DROP_COLUMNFAMILY     TIMESTAMP_INPUT
BEGIN                          DROP_INDEX            TIMESTAMP_OUTPUT
BLOB_INPUT                     DROP_KEYSPACE         TRUNCATE
BOOLEAN_INPUT                  DROP_TABLE            TYPES
CREATE                         DROP_USER             UPDATE
CREATE_COLUMNFAMILY            GRANT                 UPDATE_COUNTERS
CREATE_COLUMNFAMILY_OPTIONS    INSERT                UPDATE_SET
CREATE_COLUMNFAMILY_TYPES      LIST                  UPDATE_USING
CREATE_INDEX                   LIST_PERMISSIONS      UPDATE_WHERE
CREATE_KEYSPACE                LIST_USERS            USE
CREATE_TABLE                   PERMISSIONS           UUID_INPUT

cqlsh>
```

7.5 KEYSPACES

What is a keyspace? A keyspace is a container to hold application data. It is comparable to a relational database. It is used to group column families together. Typically, a cluster has one keyspace per application. Replication is controlled on a per keyspace basis. Therefore, data that has different replication requirements should reside on different keyspaces.

When one creates a keyspace, it is required to specify a strategy class. There are two choices available with us. Either we can specify a "SimpleStrategy" or a "NetworkTopologyStrategy" class. While using Cassandra for evaluation purpose, go with "SimpleStrategy" class and for production usage, work with the "NetworkTopologyStrategy" class.

Objective: To create a keyspace by the name "Students".

Act:

```
CREATE KEYSPACE Students WITH REPLICATION = {
        'class':'SimpleStrategy',
        'replication_factor':1
};
```

Outcome:

```
C:\windows\system32\cmd.exe - Python cqlsh
cqlsh> CREATE KEYSPACE Students WITH REPLICATION = {
   ...                 'class':'SimpleStrategy',
   ...                 'replication_factor':1
   ...  };
cqlsh>
```

The replication factor stated above in the syntax for creating keyspace is related to the number of copies of keyspace data that is housed in a cluster.

Objective: To describe all the existing keyspaces.

Act:

```
DESCRIBE KEYSPACES;
```

Outcome:

```
C:\windows\system32\cmd.exe - python cqlsh

d:\apache-cassandra-2.0.0\apache-cassandra-2.0.0\apache-cassandra-2.0.0\bin>python cq
Connected to Test Cluster at localhost:9160.
[cqlsh 4.0.0 | Cassandra 2.0.0 | CQL spec 3.1.0 | Thrift protocol 19.37.0]
Use HELP for help.
cqlsh> describe keyspaces;

system  students  system_traces

cqlsh>
```

Objective: To get more details on the existing keyspaces such as keyspace name, durable writes, strategy class, strategy options, etc.

Act:

```
SELECT *
        FROM system.schema_keyspaces;
```

Outcome:

```
cqlsh> SELECT * FROM system.schema_keyspaces;

 keyspace_name | durable_writes | strategy_class                                  | strategy_options
---------------+----------------+-------------------------------------------------+----------------------------
       demo_con |           True | org.apache.cassandra.locator.SimpleStrategy     | {"replication_factor":"3"}
         system |           True |  org.apache.cassandra.locator.LocalStrategy     |                         {}
 system_traces |           True | org.apache.cassandra.locator.SimpleStrategy     | {"replication_factor":"1"}
       students |           True | org.apache.cassandra.locator.SimpleStrategy     | {"replication_factor":"1"}

(4 rows)
```

Note: Cassandra converted the Students keyspace to lowercase as quotation marks were not used.

Objective: To use the keyspace "Students", use the following command:

Use keyspace_name

Use connects the client session to the specified keyspace.

Act:

```
USE Students;
```

Outcome:

```
C:\windows\system32\cmd.exe - python cqlsh
cqlsh> use Students;
cqlsh:students>
```

Objective: To create a column family or table by the name "student_info".

Act:

```
CREATE TABLE Student_Info (
   RollNo int PRIMARY KEY,
   StudName text,
   DateofJoining timestamp,
   LastExamPercent double
   );
```

Outcome:

```
cqlsh> use Students;
cqlsh:students> CREATE TABLE Student_Info (
            ...   RollNo int PRIMARY Key,
            ...   StudName text,
            ...   DateofJoining timestamp,
            ...   LastExamPercent double
            ...   );
```

The table "student_info" gets created in the keyspace "students".

Note: Tables can have either a single or compound primary key. Always ensure that there is exactly one primary key definition. The primary key, however, can be simple (consisting of a single attribute) or composite (comprising two or more attributes).

Explanation about the composite PRIMARY KEY:

Primary key (column_name1, *column_name2, column_name3* ...)
Primary key ((column_name4, *column_name5*), column_name6, *column_name7* ...)

In the above syntax,
column_name1 is the partition key
column_name2 and column_name3 are the clustering columns.
column_name4 and column_name5 are the partitioning keys
column_name6 and column_name7 are the clustering columns.

The partition key is used to distribute the data in the table across various nodes that constitute the cluster. The clustering columns are used to store data in ascending order on the disk.

Objective: To lookup the names of all tables in the current keyspace, or in all the keyspaces if there is no current keyspace.

Act:

```
DESCRIBE TABLES;
```

Outcome:

```
cqlsh:students> describe tables;

student_info

cqlsh:students>
```

Objective: To describe the table "student_info" use the below command.

Act:

```
DESCRIBE TABLE student_info;
```

Note: The output is a list of CQL commands with the help of which the table "student_info" can be recreated.

Outcome:

```
C:\windows\system32\cmd.exe - python cqlsh
cqlsh:students> describe table student_info;

CREATE TABLE student_info (
  rollno int,
  dateofjoining timestamp,
  lastexampercent double,
  studname text,
  PRIMARY KEY (rollno)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

7.6 CRUD (CREATE, READ, UPDATE, AND DELETE) OPERATIONS

Objective: To insert data into the column family "student_info".

An insert writes one or more columns to a record in Cassandra table atomically. An insert statement does not return an output. One is not required to place values in all the columns; however, it is mandatory to specify all the columns that make up the primary key. The columns that are missing do not occupy any space on disk.

Internally insert and update operations are equal. However, insert does not support counters but update does. Counters will be discussed later in the chapter.

Act:

```
BEGIN BATCH
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (1,'Michael Storm','2012-03-29', 69.6)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (2,'Stephen Fox','2013-02-27', 72.5)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (3,'David Flemming','2014-04-12', 81.7)
INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
VALUES (4,'Ian String','2012-05-11', 73.4)
APPLY BATCH;
```

Outcome:

```
cqlsh:students> BEGIN BATCH
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ... VALUES (1,'Michael Storm','2012-03-29',69.6)
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ... VALUES (2,'Stephen Fox','2013-02-27',72.5)
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ...  VALUES (3,'David Flemming ','2014-04-12',81.7)
           ...   INSERT INTO student_info (RollNo,StudName,DateofJoining,LastExamPercent)
           ...  VALUES (4,'Ian String','2012-05-11',73.4)
           ... APPLY BATCH;
cqlsh:students>
```

Objective: To view the data from the table "student_info".

Act:

```
SELECT *
        FROM student_info;
```

The above select statement retrieves data from the "student_info" table.

Outcome:

```
cqlsh:students> select * from student_info;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+---------------
      1 | 2012-03-29 00:00:00India Standard Time |            69.6 |   Michael Storm
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 |     Stephen Fox
      4 | 2012-05-11 00:00:00India Standard Time |            73.4 |      Ian String
      3 | 2014-04-12 00:00:00India Standard Time |            81.7 | David Flemming

(4 rows)

cqlsh:students>
```


Objective: To view only those records where the RollNo column either has a value 1 or 2 or 3.

Act:

```
SELECT *
        FROM student_info
                WHERE RollNo IN(1,2,3);
```

Note: For the above statement to execute successfully, ensure that the following criteria are satisfied:

1. Either the partition key definition includes the column that is used in the where clause i.e. search criteria.
2. OR the column being used in the where clause, that is, search criteria, has an index defined on it using the CREATE INDEX statement.

Outcome:

```
cqlsh:students> Select * from student_info where RollNo IN(1,2,3);

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+---------------
      1 | 2012-03-29 00:00:00India Standard Time |            69.6 |   Michael Storm
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 |     Stephen Fox
      3 | 2014-04-12 00:00:00India Standard Time |            81.7 | David Flemming

(3 rows)

cqlsh:students>
```

Let us try running a query with "studname" in the where clause. Since "studname" is neither the primary key column nor a column in the primary key definition and also does not have an index defined on it, such a query will lead to error.

We set the stage to resolve the error by creating an index on the "studname" column of the "student_info" table and then subsequently executing the query.

To create an index on the "studname" column of the "student_info" column family use

```
CREATE INDEX ON student_info(studname)
```

To execute the query using the index defined on "studname" column use

```
SELECT *
        FROM student_info
                WHERE studname='Stephen Fox' ;
```

Outcome:

```
cqlsh:students> create index on student_info(studname);
cqlsh:students> select * from student_info where studname='Stephen Fox' ;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 | Stephen Fox

(1 rows)

cqlsh:students>
```

Objective: Let us create another index on the "LastExamPercent" column of the "student_info" column family.

Act:

```
CREATE INDEX ON student_info(LastExamPercent);
```

Outcome:

```
cqlsh:students> create index on student_info(LastE
cqlsh:students> select * from student_info where LastExamPercent = 81.7;

 rollno | dateofjoining                             | lastexampercent | studname
--------+-------------------------------------------+-----------------+---------------
      3 | 2014-04-12 00:00:00India Standard Time |            81.7 | David Flemming

(1 rows)
```

Objective: To specify the number of rows returned in the output using limit.

Act:

```
SELECT rollno, hobbies, language, lastexampercent
      FROM student_info LIMIT 2;
```

Outcome:

```
cqlsh:students> select rollno, hobbies, language, lastexampercent from student_info limit 2;

 rollno | hobbies                                | language          | lastexampercent
--------+----------------------------------------+-------------------+-----------------
      1 |                ['Chess, Table Tennis'] | ['Hindi, English'] |            69.6
      4 | {'Lawn Tennis, Table Tennis, Golf'} | ['Hindi, English'] |            73.4

(2 rows)
```

Objective: To use column alias for the column "language" in the "student_info" table. We would like the column heading to be "knows language".

Act:

```
SELECT rollno, language AS "knows language"
      FROM student_info;
```

Outcome:

```
cqlsh:students> select rollno, language as "knows language" from student_info;

 rollno | knows language
--------+-----------------------
      1 |       ['Hindi, English']
      4 |       ['Hindi, English']
      3 | ['Hindi,English,French']

(3 rows)
```

Objective: To update the value held in the "StudName" column of the "student_info" column family to "David Sheen" for the record where the RollNo column has value = 2.

Note: An update updates one or more column values for a given row to the Cassandra table. It does not return anything.

Act:

```
UPDATE student_info SET StudName = 'David Sheen' WHERE RollNo = 2;
```

Outcome:

```
cqlsh:students> UPDATE student_info  SET StudName = 'David Sheen'  WHERE RollNo = 2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                             | lastexampercent | studname
--------+-------------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            72.5 | David Sheen

(1 rows)
cqlsh:students>
```

Objective: Let us try updating the value of a primary key column.

Act:

```
UPDATE student_info SET rollno=6 WHERE rollno=3;
```

Outcome:

```
cqlsh:students> update student_info set rollno=6 where rollno=3;
Bad Request: PRIMARY KEY part rollno found in SET part
cqlsh:students>
```

Note: It does not allow update to a primary key column.

Objective: Updating more than one column of a row of Cassandra table.

Act:

Step 1: Before the update

```
cqlsh:students> select rollno, studname, lastexampercent from student_info where rollno=3;

 rollno | studname        | lastexampercent
--------+-----------------+-----------------
      3 | David Flemming  |            81.7

(1 rows)
```

Step 2: Applying the update

```
cqlsh:students> select rollno, studname, lastexampercent from student_info where rollno=3;

 rollno | studname | lastexampercent
--------+----------+-----------------
      3 |  Samaira |              85

(1 rows)
```

Step 3: After the update

```
cqlsh:students> DELETE LastExamPercent FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                             | lastexampercent | studname
--------+-------------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            null | David Sheen

(1 rows)
cqlsh:students>
```

Objective: To delete the column "LastExamPercent" from the "student_info" table for the record where the RollNo = 2.

Note: Delete statement removes one or more columns from one or more rows of a Cassandra table or removes entire rows if no columns are specified.

Act:

```
DELETE LastExamPercent FROM student_info WHERE RollNo=2;
```

Outcome:

```
cqlsh:students> DELETE LastExamPercent FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno = 2;

 rollno | dateofjoining                           | lastexampercent | studname
--------+-----------------------------------------+-----------------+-------------
      2 | 2013-02-27 00:00:00India Standard Time |            null | David Sheen

(1 rows)

cqlsh:students>
```

Objective: To delete a row (where RollNo = 2) from the table "student_info".

Act:

```
DELETE FROM student_info WHERE RollNo=2;
```

Outcome:

```
cqlsh:students> DELETE  FROM student_info where RollNo=2;
cqlsh:students> select * from student_info where rollno=2;

(0 rows)

cqlsh:students>
```

Objective: To create a table "project _details" with primary key as (project_id, project_name).

Act:

```
CREATE TABLE project_details (
        project_id int,
        project_name text,
        stud_name text,
        rating double,
        duration int,
        PRIMARY KEY (project_id, project_name));
```


Outcome:

```
cqlsh:students> CREATE TABLE project_details (
           ...   project_id int,
           ...   project_name text,
           ...   stud_name text,
           ...   rating double,
           ...   duration int,
           ...   PRIMARY KEY (project_id, project_name));
cqlsh:students>
```

Objective: To insert data into the column family "project_details".

Act:

```
BEGIN BATCH
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (1,'MS data migration','David Sheen',3.5,720)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (1,'MS Data Warehouse','David Sheen',3.9,1440)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (2,'SAP Reporting','Stephen Fox',4.2,3000)
INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
VALUES (2,'SAP BI DW','Stephen Fox',4,9000)
APPLY BATCH;
```

Outcome:

```
cqlsh:students> BEGIN BATCH
           ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
           ... VALUES (1,'MS data Migration','David Sheen',3.5,720)
           ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
           ... VALUES (1,'MS Data Warehouse','David Sheen',3.9,1440)
           ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
           ... VALUES (2,'SAP Reporting','Stephen Fox',4.2,3000)
           ...   INSERT INTO project_details (project_id,project_name,stud_name,rating,duration)
           ... VALUES (2,'SAP BI DW','Stephen Fox',4,9000)
           ... APPLY BATCH;
```

Objective: To view all the rows of the "project_details" table.

Act:

```
SELECT *
        FROM project_details;
```

Outcome:

```
cqlsh:students> select * from project_details;

 project_id | project_name       | duration | rating | stud_name
------------+--------------------+----------+--------+-------------
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen
          1 | MS data Migration |      720 |    3.5 | David Sheen
          2 |        SAP BI DW |     9000 |      4 | Stephen Fox
          2 |     SAP Reporting |     3000 |    4.2 | Stephen Fox

(4 rows)
```

Objective: To view row/record from the "project_details" table wherein the project_id=1.

Act:

```
SELECT *
    FROM project_details
        WHERE project_id=1;
```

Outcome:

```
cqlsh:students> Select * from project_details where project_id=1;

 project_id | project_name      | duration | rating | stud_name
------------+-------------------+----------+--------+-------------
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen
          1 | MS data Migration |      720 |    3.5 | David Sheen

(2 rows)
```

Objective: To use "allow filtering" with the Select statement.

Note: When one attempts a potentially expensive query that might involve searching a range of rows, a prompt such as the one shown below appears:

> Bad Request: Cannot execute this query as it might involve data filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING.

Act:

```
SELECT *
    FROM project_details
        WHERE project_name='MS Data Warehouse' ALLOW FILTERING;
```

Outcome:

```
cqlsh:students> Select * from project_details where project_name='MS Data Warehouse' allow filtering;

 project_id | project_name      | duration | rating | stud_name
------------+-------------------+----------+--------+-------------
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen

(1 rows)
```

Objective: To sort or order the rows/records of the "project_details" column in ascending order of project_name.

Act:

```
SELECT *
    FROM project_details
        WHERE project_id IN (1,2)
            ORDER BY project_name DESC;
```


Outcome:

```
cqlsh:students> SELECT * FROM project_details WHERE project_id IN (1,2) ORDER BY project_name DESC;

 project_id | project_name      | duration | rating | stud_name
------------+-------------------+----------+--------+-------------
          2 |    SAP Reporting |     3000 |    4.2 | Stephen Fox
          2 |       SAP BI DW |     9000 |      4 | Stephen Fox
          1 | MS data Migration |      720 |    3.5 | David Sheen
          1 | MS Data Warehouse |     1440 |    3.9 | David Sheen

(4 rows)
```

Note: By default sorting or ordering is done in ascending order. The user can specify the order by using the keyword "ASC" for ascending or "DESC" for descending.

ORDER BY clause can select a single column only. This column is the second column of the compound primary key. This applies even when the compound primary key has more than two columns.

When specifying the ORDER BY clause, use only the column name and not the column alias.

7.7 COLLECTIONS

7.7.1 Set Collection

A column of type set consists of unordered unique values. However, when the column is queried, it returns the values in sorted order. For example, for text values, it sorts in alphabetical order.

PICTURE THIS...

You are required to store details about users of service "xyz". The details of the user include: User_ID, User_Name, User_Contact_Nos, User_Email_Ids. A user may have n number of Contact Nos and also may have n number of Email IDs. How do we accomplish this task in RDBMS?

We would create a table, let us say "Users", to store details such as "User_ID", "User_Name" and another table "UsersContactDetails". The relationship between "UsersContactDetails" and "Users" is many-to-one. Likewise, we would create a table "UsersEmailIDs" and establish a many-to-one relationship between "UserEmailIDs" and the "Users" table.

However, the multiple Contact Nos and multiple Email IDs problem can be solved by defining a column as a collection. The usage of collection types for columns is not only convenient but intuitive as well.

CQL makes use of the following collection types:

- Set
- List
- Map

When to use collection?
Use collection when it is required to store or denormalize a small amount of data.

What is the limit on the values of items in a collection?
The values of items in a collection are limited to 64K.

Where to use collections?
Collections can be used when you need to store the following:

1. Phone numbers of users.
2. Email ids of users.

When should one refrain from using a collection?
One should refrain from using a collection when the data has unbound growth potential such as all the messages posted by a user or all the event data as captured by a sensor. When faced with such a situation, use a table with compound primary key with data being held in clustering columns.

7.7.2 List Collection

When the order of elements matter, one should go for a list collection. For example, when you store the preferences of places to visit by a user, you would like to respect his preferences and retrieve the values in the order in which he has entered rather than in sorted order. A list also allows one to store the same value multiple times.

7.7.3 Map Collection

As the name implies, a map is used to map one thing to another. A map is a pair of typed values. It is used to store timestamp related information. Each element of the map is stored as a Cassandra column. Each element can be individually queried, modified, and deleted.

Objective: To alter the schema for the table "student_info" to add a column "hobbies".

Act:

```
ALTER TABLE student_info ADD hobbies set<text>;
```

Outcome:

```
cqlsh:students> ALTER TABLE student_info ADD hobbies set<text>;
```

Confirm the structure of the table after the change has been made:

```
cqlsh:students> describe table student_info;

CREATE TABLE student_info (
  rollno int,
  dateofjoining timestamp,
  hobbies set<text>,
  lastexampercent double,
  studname text,
  PRIMARY KEY (rollno)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

CREATE INDEX student_info_lastexampercent_idx ON student_info (lastexampercent);

CREATE INDEX student_info_studname_idx ON student_info (studname);
```

Objective: To alter the schema of the table "student_info" to add a list column "language".

Act:

```
ALTER TABLE student_info ADD language list<text>;
```


Outcome:

```
cqlsh:students> ALTER TABLE student_info ADD language list<text>;
cqlsh:students>
```

Confirm the structure of the table after the change has been made:

```
cqlsh:students> describe table student_info;

CREATE TABLE student_info (
  rollno int,
  dateofjoining timestamp,
  hobbies set<text>,
  language list<text>,
  lastexampercent double,
  studname text,
  PRIMARY KEY (rollno)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

CREATE INDEX student_info_lastexampercent_idx ON student_info (lastexampercent);

CREATE INDEX student_info_studname_idx ON student_info (studname);

cqlsh:students>
```

Objective: To update the table "student_info" to provide the values for "hobbies" for the student with Rollno =1.

Act:

```
UPDATE student_info
        SET hobbies = hobbies + {'Chess, Table Tennis'}
                WHERE RollNo=1;
```

Outcome:

```
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Chess, Table Tennis'} WHERE RollNo=1;
```

To confirm the values in the hobbies column, use the below command:

```
SELECT *
        FROM student_info
                WHERE RollNo=1;
```

```
cqlsh:students> select * from student_info where RollNo=1;
 rollno | dateofjoining                          | hobbies                | language | lastexampercent | studname
--------+----------------------------------------+------------------------+----------+-----------------+---------------
      1 | 2012-03-29 00:00:00India Standard Time | {'Chess, Table Tennis'} |     null |            69.6 | Michael Storm
(1 rows)
```

Likewise update a few more records:

```
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Chess, Badminton'} WHERE RollNo=3;
cqlsh:students>
cqlsh:students> UPDATE student_info
           ...    SET hobbies = hobbies + {'Lawn Tennis, Table Tennis, Golf'} WHERE RollNo=4;
cqlsh:students>
```

Records after the updation:

```
cqlsh:students> select * from student_info;

 rollno | dateofjoining                          | hobbies                                | language | lastexampercent | studna
--------+----------------------------------------+----------------------------------------+----------+-----------------+-------
      1 | 2012-03-29 00:00:00India Standard Time |                  {'Chess, Table Tennis'} |     null |            69.6 |   Mich
      4 | 2012-05-11 00:00:00India Standard Time | {'Lawn Tennis, Table Tennis, Golf'} |     null |            73.4 |      I
      3 | 2014-04-12 00:00:00India Standard Time |                  {'Chess, Badminton'} |     null |            81.7 | David

(3 rows)
```

Objective: To update values in the list column, "language" of the table "student_info".

Act:

```
UPDATE student_info
        SET language = language + ['Hindi, English']
                WHERE RollNo=1;
```

Outcome:

```
cqlsh:students> UPDATE student_info
           ...    SET language = language + ['Hindi, English'] WHERE RollNo=1;
cqlsh:students>
```

Likewise update the remaining records.

```
cqlsh:students> UPDATE student_info
           ...    SET language = language + ['Hindi,English,French'] WHERE RollNo=3;
cqlsh:students>
cqlsh:students> UPDATE student_info
           ...    SET language = language + ['Hindi, English'] WHERE RollNo=4;
cqlsh:students>
```

To view the updates to the records, use the below statement:

```
cqlsh:students> select rollNo, studname, hobbies, language from student_info;

 rollno | studname        | hobbies                                | language
--------+-----------------+----------------------------------------+-------------------------
      1 |   Michael Storm |                  {'Chess, Table Tennis'} |       ['Hindi, English']
      4 |      Ian String | {'Lawn Tennis, Table Tennis, Golf'} |       ['Hindi, English']
      3 | David Flemming  |                  {'Chess, Badminton'} | ['Hindi,English,French']

(3 rows)
```

7.7.4 More Practice on Collections (SET and LIST)

Objective: To create a table "users" with an "emails" column. The type of this column "emails" is "set".

Act:

```
CREATE TABLE users (
        user_id text PRIMARY KEY,
        first_name text,
        last_name text,
        emails set<text>
);
```

Outcome:

```
cqlsh:students> CREATE TABLE users (
           ...   user_id text PRIMARY KEY,
           ...   first_name text,
           ...   last_name text,
           ...   emails set<text>
           ... );
cqlsh:students>
```

Objective: To insert values into the "emails" column of the "users" table.

Note: Set values must be unique.

Act:

```
INSERT INTO users
        (user_id, first_name, last_name, emails)
                VALUES('AB', 'Albert', 'Baggins', {'a@baggins.com', 'baggins@gmail.com'});
```

Outcome:

```
cqlsh:students> INSERT INTO users (user_id, first_name, last_name, emails)
           ...   VALUES('AB', 'Albert', 'Baggins', {'a@baggins.com', 'baggins@gmail.com'});
cqlsh:students>
```

Objective: Add an element to a set using the UPDATE command and the addition (+) operator.

Act:

```
UPDATE users
        SET emails = emails + {'ab@friendsofmordor.org'}
                WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> UPDATE users
           ...   SET emails = emails + {'ab@friendsofmordor.org'} WHERE user_id = 'AB';
cqlsh:students>
```

Objective: To retrieve email addresses for Albert from the set.

Act:

```
SELECT user_id, emails
        FROM users
                WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> SELECT user_id, emails FROM users WHERE user_id = 'AB';

 user_id | emails
---------+-------------------------------------------------------------------
      AB | {'a@baggins.com', 'ab@friendsofmordor.org', 'baggins@gmail.com'}

(1 rows)
```

Objective: To remove an element from a set using the subtraction (–) operator.

Act:

```
UPDATE users
        SET emails = emails - {'ab@friendsofmordor.org'}
                WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> UPDATE users
           ...   SET emails = emails - {'ab@friendsofmordor.org'} WHERE user_id = 'AB';
```

To view the records from the "users" table:

```
cqlsh:students> select * from users;

 user_id | emails                                    | first_name | last_name
---------+-------------------------------------------+------------+-----------
      AB | {'a@baggins.com', 'baggins@gmail.com'} |     Albert |   Baggins

(1 rows)
```

Objective: To remove all elements from a set by using the UPDATE or DELETE statement.

Act:

```
UPDATE users
        SET emails = {}
                WHERE user_id = 'AB';
cqlsh:students> UPDATE users SET emails = {} WHERE user_id = 'AB';
```


Outcome: The above command removes the emails column. Here is the confirmation:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name
---------+--------+------------+-----------
      AB |   null |     Albert |   Baggins

(1 rows)
```

OR

```
DELETE emails
        FROM users
                WHERE user_id = 'AB';
cqlsh:students> DELETE emails FROM users WHERE user_id = 'AB';
```

The above command removes the emails column. Here is the confirmation:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name
---------+--------+------------+-----------
      AB |   null |     Albert |   Baggins

(1 rows)
```

Objective: To alter the "users" table to add a column, "top_places" of type list.

Act:

```
ALTER TABLE users ADD top_places list<text>;
```

```
cqlsh:students> ALTER TABLE users ADD top_places list<text>;
```

Outcome:

The above command alters the structure of the table, "users". Here is the confirmation.

```
cqlsh:students> describe table users;

CREATE TABLE users (
  user_id text,
  emails set<text>,
  first_name text,
  last_name text,
  top_places list<text>,
  PRIMARY KEY (user_id)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

Objective: To update the list column "top_places" in the "users" table for user_id = 'AB'.

Act:

```
UPDATE users
    SET top_places = [ 'Lonavla', 'Khandala' ]
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users
           ...   SET top_places = [ 'Lonavla', 'Khandala' ] WHERE user_id = 'AB';
cqlsh:students>
```

Outcome:

```
cqlsh:students> select * from users where user_id = 'AB';

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-------------------------
      AB |   null |     Albert |   Baggins | ['Lonavla', 'Khandala']

(1 rows)
```

Objective: Prepend an element to the list by enclosing it in square brackets and using the addition (+) operator.

Act:

```
UPDATE users
    SET top_places = [ 'Mahabaleshwar' ] + top_places
        WHERE user_id = 'AB';
```

```
cqlsh:students> UPDATE users
           ...   SET top_places = [ 'Mahabaleshwar' ] + top_places WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)
```

Objective: To append an element to the list by switching the order of the new element data and the list name in the update command.

Act:

```
UPDATE users
    SET top_places = top_places + [ 'Tapola' ]
        WHERE user_id = 'AB';
```



```
cqlsh:students> UPDATE users
           ...   SET top_places = top_places + [ 'Tapola' ] WHERE user_id = 'AB';
cqlsh:students>
```

Outcome:

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Objective: To query the database for a list of top places.

Act:

```
SELECT user_id, top_places
    FROM users
        WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> SELECT user_id, top_places FROM users WHERE user_id = 'AB';

 user_id | top_places
---------+-----------------------------------------------------
      AB | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Objective: To remove an element from a list using the DELETE command and the list index position in square brackets.

The record as it exists prior to deletion is

```
cqlsh:students> SELECT user_id, top_places FROM users WHERE user_id = 'AB';

 user_id | top_places
---------+-----------------------------------------------------
      AB | ['Mahabaleshwar', 'Lonavla', 'Khandala', 'Tapola']

(1 rows)
```

Act:

```
DELETE top_places[3]
    FROM users
        WHERE user_id = 'AB';
```

```
cqlsh:students> DELETE top_places[3] FROM users WHERE user_id = 'AB';
```

Outcome: The status after deletion is

```
cqlsh:students> select * from users;

 user_id | emails | first_name | last_name | top_places
---------+--------+------------+-----------+-----------------------------------------------
      AB |   null |     Albert |   Baggins | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)
```

7.7.5 Using Map: Key, Value Pair

Objective: To alter the "users" table to add a map column "todo".

Act:

```
ALTER TABLE users
        ADD todo map<timestamp, text>;
```

```
cqlsh:students> ALTER TABLE users ADD todo map<timestamp, text>;
```

Outcome:

```
cqlsh:students> describe table users;

CREATE TABLE users (
  user_id text,
  emails set<text>,
  first_name text,
  last_name text,
  todo map<timestamp, text>,
  top_places list<text>,
  PRIMARY KEY (user_id)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

cqlsh:students>
```

Objective: To update the record for user (user_id = 'AB') in the "users" table.

Act: The record from user_id = 'AB' as it exists in the "users" table is

```
cqlsh:students> select * from users where user_id='AB';

 user_id | emails | first_name | last_name | todo | top_places
---------+--------+------------+-----------+------+-----------------------------------------
      AB |   null |     Albert |   Baggins | null | ['Mahabaleshwar', 'Lonavla', 'Khandala']

(1 rows)

cqlsh:students> UPDATE users
            ...   SET todo =
            ...   {'2014-9-24':'Cassandra Session',
            ...    '2014-10-2 12:00' :'MongoDB Session'}
            ...   WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select user_id, todo from users where user_id='AB';

r_id | todo
-----+-----------------------------------------------------------------------------------------------------------------------
  AB | {'2014-09-24 00:00:00India Standard Time': 'Cassandra Session', '2014-10-02 12:00:00India Standard Time': 'MongoDB Se
```


Objective: To delete an element from the map using the DELETE command and enclosing the timestamp of the element in square brackets.

Act:

```
DELETE todo['2014-9-24']
        FROM users
                WHERE user_id = 'AB';
```

```
cqlsh:students> DELETE todo['2014-9-24'] FROM users WHERE user_id = 'AB';
```

Outcome:

```
cqlsh:students> select user_id, todo from users where user_id='AB';

 user_id | todo
---------+-----------------------------------------------------------------
      AB | {'2014-10-02 12:00:00India Standard Time': 'MongoDB Session'}

(1 rows)
```

7.8 USING A COUNTER

A counter is a special column that is changed in increments. For example, we may need a counter column to count the number of times a particular book is issued from the library by the student.

Step 1:

```
CREATE TABLE library_book (
        counter_value counter,
        book_name varchar,
        stud_name varchar,
        PRIMARY KEY (book_name, stud_name)
    );
```

```
cqlsh:students> CREATE TABLE library_book
            ...   (     counter_value counter,
            ...         book_name varchar,
            ...         stud_name varchar,
            ...   PRIMARY KEY (book_name, stud_name)
            ... );
```

Step 2: Load data into the counter column.

```
UPDATE library_book
        SET counter_value = counter_value + 1
                WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

```
cqlsh:students> UPDATE library_book
            ...  SET counter_value = counter_value + 1
            ... WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

Step 3: Take a look at the counter value.

SELECT *

FROM library_book;

Output is:

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             1

(1 rows)
```

Step 4: Let us increase the value of the counter.

UPDATE library_book

SET counter_value = counter_value + 1

WHERE book_name='Fundamentals of Business Analytics' AND stud_name='shaan';

```
cqlsh:students> UPDATE library_book
           ...  SET counter_value = counter_value + 1
           ... WHERE book_name='Fundamentals of Business Analytics' AND stud_name='shaan';
```

Step 5: Again, take a look at the counter value.

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             1
 Fundamentals of Business Analytics |     shaan |             1

(2 rows)
```

Step 6: Update another record for Stud_name "Jeet".

UPDATE library_book

SET counter_value = counter_value + 1

WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';

```
cqlsh:students> UPDATE library_book
           ...  SET counter_value = counter_value + 1
           ... WHERE book_name='Fundamentals of Business Analytics' AND stud_name='jeet';
```

Step 7: Let us take a look at the counter value, one last time.

```
cqlsh:students> select * from library_book;

 book_name                          | stud_name | counter_value
------------------------------------+-----------+---------------
 Fundamentals of Business Analytics |      jeet |             2
 Fundamentals of Business Analytics |     shaan |             1

(2 rows)
```

7.9 TIME TO LIVE (TTL)

Data in a column, other than a counter column, can have an optional expiration period called TTL (time to live). The client request may specify a TTL value for the data. The TTL is specified in seconds.

CREATE TABLE userlogin(

userid int primary key, password text

);


```
cqlsh:students> CREATE TABLE  userlogin(
           ... userid int primary key, password text
           ... );
```

INSERT INTO userlogin (userid, password)

VALUES (1,'infy') USING TTL 30;

```
cqlsh:students> INSERT INTO userlogin (userid, password)
           ... VALUES (1,'infy') USING TTL 30;
```

SELECT TTL (password)

FROM userlogin

WHERE userid=1;

```
cqlsh:students> SELECT TTL (password)
           ... FROM userlogin
           ... WHERE userid=1;

 ttl(password)
---------------
            18

(1 rows)
```

7.10 ALTER COMMANDS

Let us look at a few Alter commands to bring about changes to the structure of the table/column family.

1. Create a table "sample" with columns "sample_id" and "sample_name".

CREATE TABLE sample(

sample_id text,

sample_name text,

PRIMARY KEY (sample_id)

);

```
cqlsh:students> Create table sample (sample_id text, sample_name text, primary key (sample_id));
cqlsh:students>
```

2. Insert a record into the table "sample".

INSERT INTO sample(

sample_id, sample_name)

VALUES ('S101', 'Big Data');

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 'S101', 'Big Data' );
```

3. View the records of the table "sample".

SELECT *

FROM sample;

```
cqlsh:students> select * from sample;

 sample_id | sample_name
-----------+-------------
      S101 |    Big Data

(1 rows)
```

7.10.1 Alter Table to Change the Data Type of a Column

1. Alter the schema of the table "sample". Change the data type of the column "sample_id" to integer from text.

ALTER TABLE sample
ALTER sample_id TYPE int;

```
cqlsh:students> alter table sample alter sample_id type int;
```

2. After the data type of the column "sample_id" is changed from text to integer, try inserting a record as follows and observe the error message:

INSERT INTO sample(sample_id, sample_name)
VALUES('S102', 'Big Data');

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 'S102', 'Big Data' );
Bad Request: Invalid STRING constant (S102) for sample_id of type int
cqlsh:students>
```

3. Try inserting a record as given below into the table "sample".

INSERT INTO sample(sample_id, sample_name)
VALUES(102, 'Big Data');

```
cqlsh:students> Insert into sample(sample_id, sample_name) values( 102, 'Big Data' );
cqlsh:students> select * from sample;

 sample_id  | sample_name
------------+-------------
 1395732529 |    Big Data
        102 |    Big Data

(2 rows)
```

4. Alter the data type of the "sample_id" column to varchar from integer.

ALTER TABLE sample
ALTER sample_id TYPE varchar;

```
cqlsh:students> alter table sample alter sample_id type varchar;
```

5. Check the records after the data type of "sample_id" has been changed to varchar from integer.

```
cqlsh:students> select * from sample;

 sample_id      | sample_name
----------------+-------------
           S101 |    Big Data
 \x00\x00\x00f |    Big Data

(2 rows)
```

7.10.2 Alter Table to Delete a Column

1. Drop the column "sample_id" from the table "sample".

ALTER TABLE sample
DROP sample_id;

```
cqlsh:students> alter table sample drop sample_id;
Bad Request: Cannot drop PRIMARY KEY part sample_id
```

Note: The request to drop the "sample_id" column from the table "sample" does not succeed as it is the primary key column.

2. Drop the column "sample_name" from the table "sample".

ALTER TABLE sample
DROP sample_name;

```
cqlsh:students> alter table sample drop sample_name;
```

Note: the above request to drop the column "sample_name" from table "sample" succeeds.

7.10.3 Drop a Table

1. Drop the column family/table "sample".

DROP columnfamily sample;

```
cqlsh:students> drop columnfamily sample;
```

The above request succeeds. The table/column family no longer exists in the keyspace.

2. Confirm the non-existence of the table "sample" in the keyspace by giving the following command:

```
cqlsh:students> describe table sample;

Column family 'sample' not found
```

7.10.4 Drop a Database

1. Drop the keyspace "students".

DROP keyspace students;

```
cqlsh:students> drop keyspace students;
```

2. Confirm the non-existence of the keyspace "students" by issuing the following command:

```
cqlsh:students> describe keyspace students;

Keyspace 'students' not found.
```

7.11 IMPORT AND EXPORT

7.11.1 Export to CSV

Objective: Export the contents of the table/column family "elearninglists" present in the "students" database to a CSV file (d:\elearninglists.csv).

Act:

Step 1: Check the records of the table "elearninglists" present in the "students" database.

SELECT *
FROM elearninglists;

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Step 2: Execute the below command at the cqlsh prompt:

COPY elearninglists (id, course_order, course_id, courseowner, title) TO 'd:\elearninglists.csv';

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) to 'd:\elearninglists.csv';
4 rows exported in 0.000 seconds.
cqlsh:students>
```

Step 3: Check the existence of the "elearninglists.csv" file in "D:\". Given below is the content of the "d:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

7.11.2 Import from CSV

Objective: To import data from "D:\elearninglists.csv" into the table "elearninglists" present in the "students" database.

Step 1: Check for the table "elearninglists" in the "students" database. If the table is already present, truncate the table. This will remove all records from the table but retain the structure of the table. In our case, the table "elearninglists" is already present in the "students" database. Let us take a look at the records of the "elearninglists" before we run the truncate command on it.

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```

Truncate the table using the below command:

TRUNCATE elearninglists;

```
cqlsh:students> Truncate  elearninglists;
cqlsh:students>
```

Note: No record is present in the table "elearninglists". The structure/schema is however preserved. We confirm it by executing the below command at the cqlsh prompt.

```
cqlsh:students> select * from elearninglists;

(0 rows)

cqlsh:students>
```

Step 2: Check for the content of the "D:\elearninglists.csv" file.

	A	B	C	D	E
1	101	1	1001	Subhashini	NoSQL Cassandra
2	101	2	1002	Seema	NoSQL MongoDB
3	101	3	1003	Seema	Hadoop Sqoop
4	101	4	1004	Subhashini	Hadoop Flume

Note: The content in the CSV agrees with the structure of the table "elearninglists" in the "students" database. The structure should be such that the content from the CSV can be housed within it without any issues.

Step 3: Execute the below command to import data from "d:\elearninglists.csv" into the table "elearninglists" in the database "students".

COPY elearninglists (id, course_order, course_id, courseowner, title) FROM 'd:\elearninglists.csv';

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) from 'd:\elearninglists.csv';
4 rows imported in 0.031 seconds.
cqlsh:students>
```

Step 4: Confirm that records have been imported into the table.

SELECT *
FROM elearninglists;

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)

cqlsh:students>
```

7.11.3 Import from STDIN

Objective: To import data into an existing table "persons" present in the "students" database. The data is to be provided by the user using the standard input device.

Step 1: Ensure that the table "persons" exists in the database "students".

DESCRIBE TABLE persons;

```
cqlsh:students> describe table persons;

CREATE TABLE persons (
  id int,
  fname text,
  lname text,
  PRIMARY KEY (id)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

cqlsh:students>
```

Step 2:

COPY persons (id, fname, lname) FROM STDIN;

```
cqlsh:students> COPY persons (id, fname, lname) FROM STDIN;
[Use \. on a line by itself to end input]
[copy] 1,"Samuel","Jones"
[copy] 2,"Virat","Kumar"
[copy] 3,"Andrew","Simon"
[copy] 4,"Raul","A Simpson"
[copy] \.

4 rows imported in 1 minute and 24.336 seconds.
cqlsh:students>
```

Step 3: Confirm that the records from the standard input device are loaded into the "persons" table existing in the "students" database.

SELECT *
FROM persons;

```
cqlsh:students> select * from persons;

 id | fname  | lname
----+--------+-----------
  1 | Samuel |     Jones
  2 |  Virat |     Kumar
  4 |   Raul | A Simpson
  3 | Andrew |     Simon

(4 rows)

cqlsh:students>
```

7.11.4 Export to STDOUT

Objective: Export the contents of the table/column family "elearninglists" present in the "students" database to the standard output device (STDOUT).

Act:

Step 1: Check the records of the table "elearninglists" present in the "students" database.

SELECT *
FROM elearninglists;

```
cqlsh:students> select * from elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            3 |      1003 |       Seema |    Hadoop Sqoop
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(4 rows)
```


Step 2: Execute the below command at the cqlsh prompt.

COPY elearninglists (id, course_order, course_id, courseowner, title) TO STDOUT;

```
cqlsh:students> copy elearninglists (id, course_order, course_id, courseowner, title) to STDOUT;
101,1,1001,Subhashini,NoSQL Cassandra
101,2,1002,Seema,NoSQL MongoDB
101,3,1003,Seema,Hadoop Sqoop
101,4,1004,Subhashini,Hadoop Flume
4 rows exported in 0.031 seconds.
cqlsh:students>
```

7.12 QUERYING SYSTEM TABLES

There are quite a few system tables such as schema_keyspaces, schema_columnfamilies, schema_columns, local, peers, etc. Let us look at what each of these system tables store in them.

SELECT *
FROM system.schema_keyspaces;

```
cqlsh:system> describe table system.schema_keyspaces;

CREATE TABLE schema_keyspaces (
  keyspace_name text,
  durable_writes boolean,
  strategy_class text,
  strategy_options text,
  PRIMARY KEY (keyspace_name)
) WITH COMPACT STORAGE AND
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='keyspace definitions' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=8640 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

SELECT *
FROM system.schema_columnfamilies;

```
CREATE TABLE schema_columnfamilies (
  keyspace_name text,
  columnfamily_name text,
  bloom_filter_fp_chance double,
  caching text,
  column_aliases text,
  comment text,
  compaction_strategy_class text,
  compaction_strategy_options text,
  comparator text,
  compression_parameters text,
  default_time_to_live int,
  default_validator text,
```

```
  dropped_columns map<text, bigint>,
  gc_grace_seconds int,
  index_interval int,
  key_aliases text,
  key_validator text,
  local_read_repair_chance double,
  max_compaction_threshold int,
  memtable_flush_period_in_ms int,
  min_compaction_threshold int,
  populate_io_cache_on_flush boolean,
  read_repair_chance double,
  replicate_on_write boolean,
  speculative_retry text,
  subcomparator text,
  type text,
  value_alias text,
  PRIMARY KEY (keyspace_name, columnfamily_name)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='ColumnFamily definitions' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=8640 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

SELECT *

FROM system.schema_columns;

```
cqlsh:system> describe table system.schema_columns;

CREATE TABLE schema_columns (
  keyspace_name text,
  columnfamily_name text,
  column_name text,
  component_index int,
  index_name text,
  index_options text,
  index_type text,
  type text,
  validator text,
  PRIMARY KEY (keyspace_name, columnfamily_name, column_name)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='ColumnFamily column attributes' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=8640 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```


SELECT *

FROM system.local;

```
CREATE TABLE local (
  key text,
  bootstrapped text,
  cluster_name text,
  cql_version text,
  data_center text,
  gossip_generation int,
  host_id uuid,
  native_protocol_version text,
  partitioner text,
  rack text,
  release_version text,
  schema_version uuid,
  thrift_version text,
  tokens set<text>,
  truncated_at map<uuid, blob>,
  PRIMARY KEY (key)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='information about the local node' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=0 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

SELECT *

FROM system.peers;

```
CREATE TABLE peers (
  peer inet,
  data_center text,
  host_id uuid,
  preferred_ip inet,
  rack text,
  release_version text,
  rpc_address inet,
  schema_version uuid,
  tokens set<text>,
  PRIMARY KEY (peer)
) WITH
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='known peers in the cluster' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=0 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};
```

7.13 PRACTICE EXAMPLES

Objective: To create table "elearninglist" with columns: id, course_order, course_id, title, courseowner.

Act:

```
CREATE TABLE elearninglists (
        id int,
        course_order int,
        course_id int,
        title text,
        courseowner text,
        PRIMARY KEY (id, course_order )
        );
```

Here, id ==> Partition Key, course_order ==> Clustering Column. The combination of the id and course_order in the elearninglists table uniquely identifies a row in the elearninglists table. You can have more than one row with the same id as long as the rows contain different course_order values.

Outcome:

```
cqlsh:students> CREATE TABLE elearninglists (
           ...   id int,
           ...   course_order int,
           ...   course_id int,
           ...   title text,
           ...   courseowner text,
           ...   PRIMARY KEY  (id, course_order ) );
cqlsh:students>
```

Objective: To insert rows into the table "elearninglists".

Act:

```
INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 1, 1001,'NoSQL Cassandra','Subhashini');

INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 2, 1002,'NoSQL MongoDB','Seema');

INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 3, 1003,'Hadoop Sqoop','Seema');

INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
VALUES (101, 4, 1004,'Hadoop Flume', 'Subhashini');
```


Outcome:

```
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101,1,1001,'NoSQL Cassandra','Subhashini');
cqlsh:students>
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101,2,1002,'NoSQL MongoDB','Seema');
cqlsh:students>
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101,3,1003,'Hadoop Sqoop','Seema',);
Bad Request: line 2:44 no viable alternative at input ')'
cqlsh:students>
cqlsh:students> INSERT INTO elearninglists (id, course_order, course_id, title, courseowner)
           ...   VALUES (101, 4,1004,'Hadoop Flume', 'Subhashini');
cqlsh:students>
```

Objective: To query the table "elearninglists".

Act:

```
SELECT *
        FROM elearninglists;
```

Outcome:

```
cqlsh:students> SELECT * FROM elearninglists;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra
 101 |            2 |      1002 |       Seema |   NoSQL MongoDB
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume

(3 rows)
```

Objective: To query the "elearninglist" table on "courseowner" as a filter.

Act:

```
SELECT *
        FROM elearninglists
                WHERE courseowner = 'Seema';
```

Outcome: The query returns an error stating "No indexed columns present in by-columns clause with Equal operator".

```
cqlsh:students> SELECT * FROM elearninglists WHERE courseowner = 'Seema';
Bad Request: No indexed columns present in by-columns clause with Equal operator
cqlsh:students>
```

Solution: Create an index on courseowner column.

```
CREATE INDEX ON
        Elearninglists(courseowner);
```

```
cqlsh:students> CREATE INDEX ON elearninglists(courseowner);
cqlsh:students>
```

Executing the same query now shows the record:

```
cqlsh:students> SELECT * FROM elearninglists WHERE courseowner = 'Seema';

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+---------------
 101 |            2 |      1002 |       Seema | NoSQL MongoDB

(1 rows)
cqlsh:students>
```

Objective: To order all the rows of elearninglists with id = 101 in descending order of "course_order". The maximum number of records to retrieve is 50.

Act:

```
SELECT *
        FROM elearninglists
                WHERE id = 101
                        ORDER BY course_order DESC LIMIT 50;
```

Outcome:

```
cqlsh:students> SELECT * FROM elearninglists WHERE id = 101 ORDER BY course_order DESC LIMIT 50;

 id  | course_order | course_id | courseowner | title
-----+--------------+-----------+-------------+-----------------
 101 |            4 |      1004 |  Subhashini |    Hadoop Flume
 101 |            2 |      1002 |       Seema |  NoSQL MongoDB
 101 |            1 |      1001 |  Subhashini | NoSQL Cassandra

(3 rows)
```

REMIND ME

- 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.
- 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.
- A replication strategy is employed to determine which nodes to place the data on. Two replication strategies are available:
  - SimpleStrategy.
  - NetworkToplogyStrategy.
- One of the features of Cassandra that has made it immensely popular is its ability to utilize tunable consistency. The database systems can go for either strong consistency or eventual consistency.
- Read consistency means how many replicas must respond before sending out the result to the client application.
- Write consistency means on how many replicas write must succeed before sending out an acknowledgement to the client application.

POINT ME (BOOK)

- ***Cassandra:*** The Definitive Guide By Eben Hewitt, Publisher: O'Reilly Media.

CONNECT ME (INTERNET RESOURCES)

- http://www.datastax.com/documentation/cassandra/2.0/cassandra/gettingStartedCassandraIntro.html
- http://www.datastax.com/documentation/cql/3.1/pdf/cql31.pdf
- http://www.datastax.com/documentation/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html
- http://www.datastax.com/docs/1.0/cluster_architecture/about_client_requests
- http://www.datastax.com/docs/datastax_enterprise3.1/solutions/about_pig

TEST ME

A. Unsolved Exercises

1. What is Cassandra?
2. Comment on Cassandra writes.
3. What is your understanding of tunable consistency?
4. What are collections in CQLSH? Where are they used?
5. Explain hinted handoffs.
6. What is Cassandra – cli?
7. Explain Cassandra's data model.
8. Explain the replication strategy in Cassandra.
9. Cassandra adheres to the Availability and Partition tolerant traits as stated by the CAP theorem. Explain.

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1: COLLECTIONS

Objective: To learn about the various collection types: Set, List and Map.
Problem Description: Design a table/column family to support the following requirements.

- Store the basic information about students such as Student Roll No, Student Name, Student Date of Birth, and Student Address.
- Store the subject preferences of each student. There should be a minimum of two subject preferences and a maximum of four. The order of preferences as given by the student should be preserved.
- Store the hobbies of each student. There should be a minimum of two hobbies and a maximum of four. The hobbies as given by the student should be arranged in alphabetical order.

ASSIGNMENT 2: TIME TO LIVE

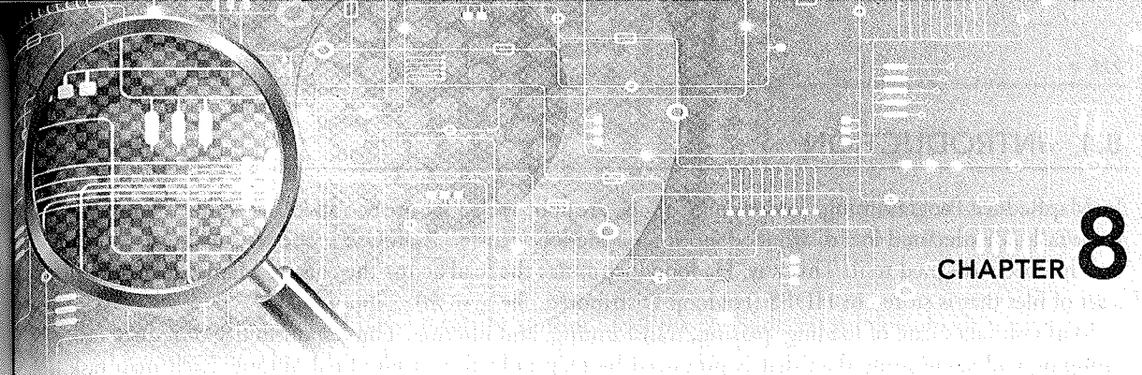
Objective: To learn about the TTL type (Time To Live).
Problem Description: Design a table/column family to support the following requirements.

Store the login details of the user such as UserID and Password. The information stored should expire in a day's time.

ASSIGNMENT 3: IMPORT FROM CSV

Objective: To learn about the import from CSV to Cassandra table/column family.
Problem Description: Read a public dataset from the site www.kdnuggets.com. If not already in CSV format, first convert to CSV format and then import into a Cassandra table/column family by the name "PublicDataSet" in the "Sample" database.

Confirm the presence of data in the table "PublicDataSet" in the "Sample" database.



CHAPTER 8

Introduction to MAPREDUCE Programming

BRIEF CONTENTS

"The alchemists in their search for gold discovered many other things of greater value."

– Arthur Schopenhauer, German Philosopher

WHAT'S IN STORE?

We assume that you are familiar with the basic concepts of HDFS and MapReduce Programming discussed in Chapters 4 and 5. The focus of this chapter will be to build on this knowledge to understand optimization techniques of MapReduce Programming such as combiner, partitioner, and compression. We will also discuss how to write MapReduce Programming for sorting and searching.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning and comprehension.

8.1 INTRODUCTION

In MapReduce Programming, Jobs (Applications) are split into a set of map tasks and reduce tasks. Then these tasks are executed in a distributed fashion on Hadoop cluster. Each task processes small subset of data that has been assigned to it. This way, Hadoop distributes the load across the cluster. MapReduce job takes a set of files that is stored in HDFS (Hadoop Distributed File System) as input.

Map task takes care of loading, parsing, transforming, and filtering. The responsibility of reduce task is grouping and aggregating data that is produced by map tasks to generate final output. Each map task is broken into the following phases:

1. RecordReader.
2. Mapper.
3. Combiner.
4. Partitioner.

The output produced by map task is known as intermediate keys and values. These intermediate keys and values are sent to reducer. The reduce tasks are broken into the following phases:

1. Shuffle.
2. Sort.
3. Reducer.
4. Output Format.

Hadoop assigns map tasks to the DataNode where the actual data to be processed resides. This way, Hadoop ensures data locality. Data locality means that data is not moved over network; only computational code is moved to process data which saves network bandwidth.

8.2 MAPPER

A mapper maps the input key–value pairs into a set of intermediate key–value pairs. Maps are individual tasks that have the responsibility of transforming input records into intermediate key–value pairs.

1. **RecordReader:** RecordReader converts a byte-oriented view of the input (as generated by the InputSplit) into a record-oriented view and presents it to the Mapper tasks. It presents the tasks with keys and values. Generally the key is the positional information and value is a chunk of data that constitutes the record.
2. **Map:** Map function works on the key–value pair produced by RecordReader and generates zero or more intermediate key–value pairs. The MapReduce decides the key–value pair based on the context.
3. **Combiner:** It is an optional function but provides high performance in terms of network bandwidth and disk space. It takes intermediate key–value pair provided by mapper and applies user-specific aggregate function to only that mapper. It is also known as local reducer.
4. **Partitioner:** The partitioner takes the intermediate key–value pairs produced by the mapper, splits them into shard, and sends the shard to the particular reducer as per the user-specific code. Usually, the key with same values goes to the same reducer. The partitioned data of each map task is written to the local disk of that machine and pulled by the respective reducer.

8.3 REDUCER

The primary chore of the Reducer is to reduce a set of intermediate values (the ones that share a common key) to a smaller set of values. The Reducer has three primary phases: Shuffle and Sort, Reduce, and Output Format.

1. **Shuffle and Sort:** This phase takes the output of all the partitioners and downloads them into the local machine where the reducer is running. Then these individual data pipes are sorted by keys which produce larger data list. The main purpose of this sort is grouping similar words so that their values can be easily iterated over by the reduce task.
2. **Reduce:** The reducer takes the grouped data produced by the shuffle and sort phase, applies reduce function, and processes one group at a time. The reduce function iterates all the values associated with that key. Reducer function provides various operations such as aggregation, filtering, and combining data. Once it is done, the output (zero or more key–value pairs) of reducer is sent to the output format.
3. **Output Format:** The output format separates key–value pair with tab (default) and writes it out to a file using record writer.

Figure 8.1 describes the chores of Mapper, Combiner, Partitioner, and Reducer for the word count problem. The Word Count problem has been discussed under "Combiner" and "Partitioner".

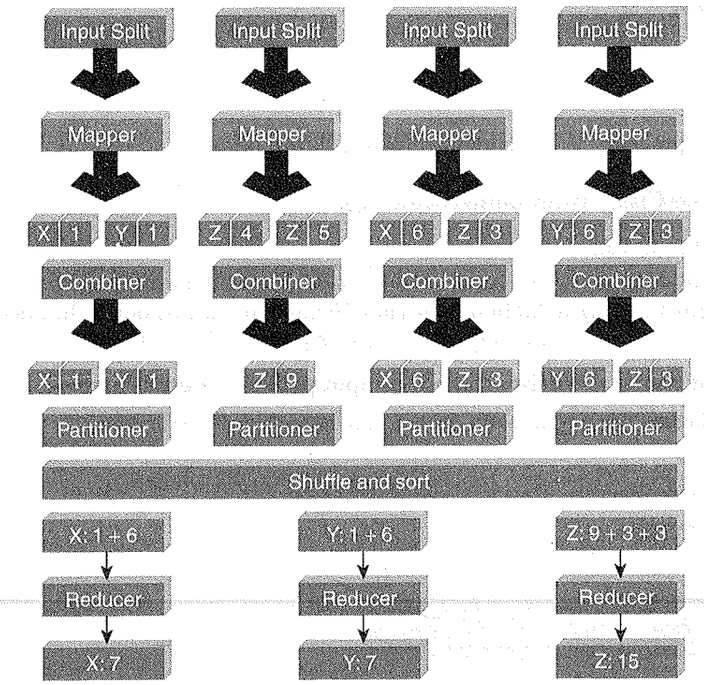


Figure 8.1 The chores of Mapper, Combiner, Partitioner, and Reducer.

8.4 COMBINER

It is an optimization technique for MapReduce Job. Generally, the reducer class is set to be the combiner class. The difference between combiner class and reducer class is as follows:

1. Output generated by combiner is intermediate data and it is passed to the reducer.
2. Output of the reducer is passed to the output file on disk.

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?
Input Data: What is the input that has been given to us to act upon?
Act: The actual statement/command to accomplish the task at hand.
Output: The result/output as a consequence of executing the statement.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use combiner for optimization.

Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act: In the driver program, set the combiner class as shown below.

```
job.setCombinerClass(WordCounterRed.class);

// Input and Output Path
FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new Path("/mapreducedemos/output/wordcount/"));
```

hadoop jar <<jar name>> <<driver class>> <<input path>> <<output path>>

Here driver class name, input path, and output path are optional arguments.

Output:

```
[root@volgalnx010 mapreducedemos]# hadoop jar wordcount.jar
```

Contents of directory /mapreducedemos

Goto : /mapreducedemos go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
lines.txt	file	91 B	3	128 MB	2015-03-01 21:05	rw-r--r--	root	supergroup
output	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Contents of directory /mapreducedemos/output

Goto : /mapreducedemos/output go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
wordcount	dir				2015-03-01 23:21	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

The reducer output will be stored in part-r-00000 file by default.

Contents of directory /mapreducedemos/output/wordcount

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup
part-r-00000	file	76 B	3	128 MB	2015-03-01 23:21	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

File: /mapreducedemos/output/wordcount/part-r-00000

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
Introducing     1
Introduction    1
Pig     1
Session 3
Welcome 1
to      2
```

8.5 PARTITIONER

The partitioning phase happens after map phase and before reduce phase. Usually the number of partitions are equal to the number of reducers. The default partitioner is hash partitioner.

Objective: Write a MapReduce program to count the occurrence of similar words in a file. Use partitioner to partition key based on alphabets.

Note: Refer Chapter 5 – Hadoop for Mapper Class and Reduce Class and Driver Program.

Input Data:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act:

WordCountPartitioner.java

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

public class WordCountPartitioner extends Partitioner<Text, IntWritable> {
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
                String word = key.toString();
                char alphabet = word.toUpperCase().charAt(0);
                int partitionNumber = 0;
                switch(alphabet) {
                case 'A': partitionNumber = 1; break;
                case 'B': partitionNumber = 2; break;
                case 'C': partitionNumber = 3; break;
                case 'D': partitionNumber = 4; break;
                case 'E': partitionNumber = 5; break;
                case 'F': partitionNumber = 6; break;
                case 'G': partitionNumber = 7; break;
                case 'H': partitionNumber = 8; break;
                case 'I': partitionNumber = 9; break;
                case 'J': partitionNumber = 10; break;
                case 'K': partitionNumber = 11; break;
                case 'L': partitionNumber = 12; break;
                case 'M': partitionNumber = 13; break;
                case 'N': partitionNumber = 14; break;
                case 'O': partitionNumber = 15; break;
                case 'P': partitionNumber = 16; break;
                case 'Q': partitionNumber = 17; break;
                case 'R': partitionNumber = 18; break;
                case 'S': partitionNumber = 19; break;
                case 'T': partitionNumber = 20; break;
                case 'U': partitionNumber = 21; break;
                case 'V': partitionNumber = 22; break;
                case 'W': partitionNumber = 23; break;
                case 'X': partitionNumber = 24; break;
                case 'Y': partitionNumber = 25; break;
                case 'Z': partitionNumber = 26; break;
                default: partitionNumber = 0; break;
                }
                return partitionNumber;
        }
}
```


In the driver program, set the partitioner class as shown below:

```
job.setNumReduceTasks(27);
job.setPartitionerClass(WordCountPartitioner.class);

// Input and Output Path
FileInputFormat.addInputPath(job, new Path("/mapreducedemos/lines.txt"));
FileOutputFormat.setOutputPath(job, new Path("/mapreducedemos/output/
wordcountpartitioner/"));
```

Output:

You can see 27 partitions in the below output.

Contents of directory /mapreducedemos/output/wordcountpartitioner

Goto : /mapreducedemos/output/wo go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00000	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00001	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00002	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00003	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00004	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00005	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00006	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00007	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00008	file	16 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00009	file	29 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00010	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00011	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00012	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00013	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00014	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00015	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00016	file	6 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00017	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00018	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00019	file	10 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00020	file	5 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00021	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00022	file	0 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00023	file	10 B	3	128 MB	2015-03-01 23:40	rw-r--r--	root	supergroup
part-r-00024	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00025	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup
part-r-00026	file	0 B	3	128 MB	2015-03-01 23:39	rw-r--r--	root	supergroup

The output file part-r-00008 is associated with alphabet **'H'**.

File: /mapreducedemos/output/wordcountpartitioner/part-r-00008

Goto : /mapreducedemos/output/wo go

Go back to dir listing
Advanced view/download options

```
Hadoop  2
Hive    2
```

8.6 SEARCHING

Objective: To write a MapReduce program to search for a specific keyword in a file.

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

Act:

WordSearcher.java

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordSearcher {

        public static void main(String[] args) throws IOException,
                        InterruptedException, ClassNotFoundException {
                Configuration conf = new Configuration();
                Job job = new Job(conf);
                job.setJarByClass(WordSearcher.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                job.setMapperClass(WordSearchMapper.class);
                job.setReducerClass(WordSearchReducer.class);
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                job.setNumReduceTasks(1);
                job.getConfiguration().set("keyword", "Jack");
                FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
```



```
                FileOutputFormat.setOutputPath(job, new Path("/mapreduce/output/search"));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

WordSearchMapper.java

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class WordSearchMapper extends Mapper<LongWritable, Text, Text, Text> {

        static String keyword;
        static int pos = 0;

        protected void setup(Context context) throws IOException,
                        InterruptedException {

                Configuration configuration = context.getConfiguration();
                keyword = configuration.get("keyword");
        }

        protected void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {

                InputSplit i = context.getInputSplit(); // Get the input split for this map.
                FileSplit f = (FileSplit) i;
                String fileName = f.getPath().getName();
                Integer wordPos;
                pos++;
                if (value.toString().contains(keyword)) {
                        wordPos = value.find(keyword);
                        context.write(value, new Text(fileName + ","+ new IntWritable(pos).
toString() + ", "+ wordPos.toString()));
                }
        }
}
```

WordSearchReducer.java

```
import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordSearchReducer extends Reducer<Text, Text, Text, Text> {

        protected void reduce(Text key, Text value, Context context)
                        throws IOException, InterruptedException {

                context.write(key, value);
        }
}
```

Output:

File: /mapreduce/output/search/part-r-00000

Goto : mapreduce/output/search go

Go back to dir listing
Advanced view/download options

```
1002,Jack,39    student.csv,2, 5
```

8.7 SORTING

Objective: To write a MapReduce program to sort data by student name (value).

Input Data:

```
1001,John,45
1002,Jack,39
1003,Alex,44
1004,Smith,38
1005,Bob,33
```

Act:

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
```



```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class SortStudNames {

        public static class SortMapper extends
                        Mapper<LongWritable, Text, Text, Text> {

                protected void map(LongWritable key, Text value, Context context)
                                throws IOException, InterruptedException {

                        String[ ] token = value.toString().split(",");
                        context.write(new Text(token[1]), new Text(token[0]+" - "+token[1]));
                }
        }
    // Here, value is sorted...
        public static class SortReducer extends
                        Reducer<Text, Text, NullWritable, Text> {

                public void reduce(Text key, Iterable<Text> values, Context context)
                                throws IOException, InterruptedException {

                        for (Text details : values) {
                                context.write(NullWritable.get(), details);
                        }
                }
        }

        public static void main(String[ ] args) throws IOException,
                        InterruptedException, ClassNotFoundException {

                Configuration conf = new Configuration();
                Job job = new Job(conf);
                job.setJarByClass(SortEmpNames.class);
                job.setMapperClass(SortMapper.class);
                job.setReducerClass(SortReducer.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);
                FileInputFormat.setInputPaths(job, new Path("/mapreduce/student.csv"));
                FileOutputFormat.setOutputPath(job, new
Path("/mapreduce/output/sorted/"));
                System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

Output:

File: /mapreduce/output/search/part-r-00000

Goto : /mapreduce/output/search go

Go back to dir listing
Advanced view/download options

```
1002,Jack,39    student.csv,2, 5
```

8.8 COMPRESSION

In MapReduce programming, you can compress the MapReduce output file. Compression provides two benefits as follows:

1. Reduces the space to store files.
2. Speeds up data transfer across the network.

You can specify compression format in the Driver Program as shown below:

```
conf.setBoolean("mapred.output.compress", true);
conf.setClass("mapred.output.compression.codec", GzipCodec.class,CompressionCodec.class);
```

Here, codec is the implementation of a compression and decompression algorithm. GzipCodec is the compression algorithm for gzip. This compresses the output file.

REMIND ME

- Mapper maps the input key–value pairs to intermediate key–value pairs.
- Reducer then reduces the set of key–value pairs that share a common key to a smaller set of values.
- The Reducer has three primary phases:
  - Shuffle and Sort
  - Reduce
  - Output Format
- Combiner and Partitioner are optimization techniques.

POINT ME (BOOK)

- MapReduce Design Patterns, O'REILLY, Donald Miner and Adam Shook.

CONNECT ME (INTERNET RESOURCES)

- http://hadooptutorial.wikispaces.com/MapReduce
- http://bigdataanalyticsnews.com/anatomy-mapreduce-job/
- http://bigdataconsultants.blogspot.in/2013/11/secondary-sort-in-hadoop-actor.html

TEST ME

A. Fill Me

1. Partitioner phase belongs to _____ task.
2. Combiner is also known as _____.
3. RecordReader converts byte-oriented view into _____ view.
4. MapReduce sorts the intermediate value based on _____.
5. In MapReduce Programming, reduce function is applied _____ group at a time.

Answers:

1. map
2. local reducer
3. record-oriented
4. keys
5. one

ASSIGNMENT FOR HANDS-ON PRACTICE

ASSIGNMENT 1

Objective: To learn about MapReduce Programming using Java.
Problem Description: Write a MapReduce Program to arrange the data on user id, then within the user id sort them in increasing order of the page count.

Input:

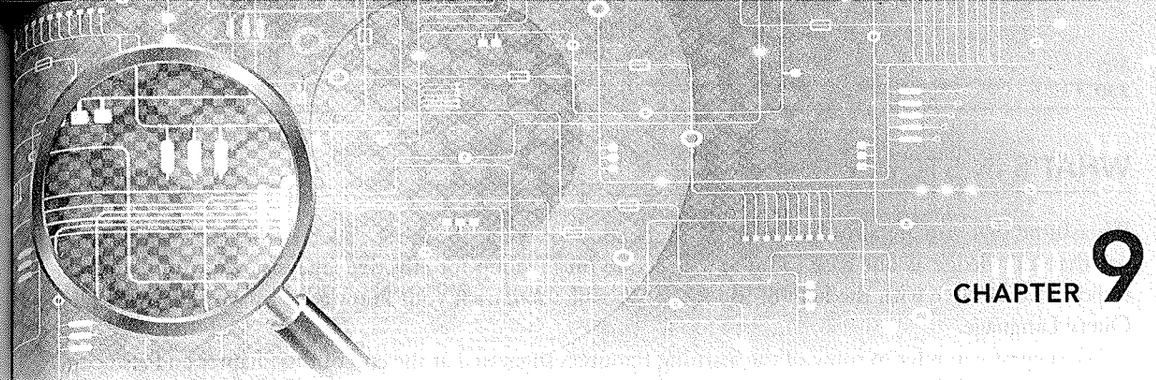
User_id	count	URL
12398	5	http://www.cbtnuggets.com/
23487	9	http://www.xda-developers.com/
34576	3	http://www.w3schools.com/
45665	6	https://www.google.co.in/
56754	4	http://www.encyclopedia.com/
67843	6	http://tutorialspoint.com/
78932	7	http://stackoverflow.com/
89021	3	http://www.wikipedia.org/
91210	2	http://www.cisce.org/results
82391	4	http://www.slideshare.net/

ASSIGNMENT 2

Objective: To learn about MapReduce Programming using Java.
Problem Description: Write a MapReduce Program to find unitwise salary.

Input:

Empno	Empname	Unit	Designation	Salary	Location
1001	John	IMST	TA	30000	Trivandrum
1002	Jack	CLOUD	PM	80000	Bangalore
1003	Joshi	FNPR	TA	35000	Trivandrum
1004	Josh	ECSSAP	PM	75000	Bangalore
1005	Jim	FSADM	SPM	60000	Bangalore
1006	Smith	ICS	TA	24000	Chandigarh
1007	Tiger	IMST	SPM	56000	Trivandrum
1008	Katé	FNPR	PM	76000	Chennai
1009	Cassy	MFGADM	TA	40000	Bangalore
1010	Ronald	ECSSAP	SPM	65000	Chennai



CHAPTER 9

Introduction to Hive

BRIEF CONTENTS

- What's in Store?
- What is Hive?
  - History of Hive and Recent Releases of Hive
  - Hive Features
  - Hive Integration and Work Flow
  - Hive Data Units
- Hive Architecture
- Hive Data Types
  - Primitive Data Types
  - Collection Data Types
- Hive File Format
  - Text File
  - Sequential File
  - RCFile (Record Columnar File)
- Hive Query Language (HQL)
  - DDL (Data Definition Language) Statements
  - DML (Data Manipulation Language) Statements
  - Starting Hive Shell
  - Database
  - Tables
  - Partitions
  - Bucketing
  - Views
  - Sub-Query
  - Joins
  - Aggregation
  - Group By and Having
- RCFILE Implementation
- SERDE
- User-Defined Function (UDF)

"Information is the oil of the 21st century, and analytics is the combustion engine."

– Peter Sondergaard, Gartner Research

WHAT'S IN STORE?

We assume that you are already familiar with commercial database systems. In this chapter, we will try to use that knowledge as our base to build a structure on Hadoop for effective analysis. We will discuss the importance of Hive with the help of use cases. We will also enrich your knowledge by working with Hive Query Language.

We suggest you refer to some of the learning resources suggested at the end of this chapter and also complete the "Test Me" exercises.

CASE STUDY: RETAIL LOG PROCESSING

About the Company

TENTOTEN is a Retail Store which has a chain of hypermarkets in India. They have 250+ stores across 95 cities and towns. About 45,000+ people are working in **TENTOTEN**. **TENTOTEN** deals in a wide range of products including fashion apparels, food products, books, furniture, etc. Around 1500+ customers visit and/or purchase products every day from each of these stores.

Problem Scenario

The approximate size of **TENTOTEN** log datasets is 12 TB. Information about the various stores is stored in the form of semi-structured data. Traditional Business Intelligence (BI) tools are good when data is present in pre-defined schema and datasets are just several hundreds of gigabytes. But the **TENTOTEN** dataset is mostly log dataset, which does not conform to any particular schema. Querying such large dataset is difficult and immensely time consuming.

The challenges are:

1. Moving the log dataset to HDFS (Hadoop Distributed File System).
2. Performing analysis on HDFS data.

Hadoop MapReduce can be used to resolve these issues. However we will still have to deal with the below constraints:

1. Writing complex MapReduce jobs in Java can be tedious and error prone.
2. Joining across large datasets is quite tricky.

Enter Hive to counter the above challenges.

9.1 WHAT IS HIVE?

Hive is a Data Warehousing tool. Refer Figure 9.1. Hive is used to query structured data built on top of Hadoop. Facebook created Hive component to manage their ever-growing volumes of log data. Hive makes use of the following:

1. HDFS for Storage.
2. MapReduce for execution.
3. Stores metadata in an RDBMS.

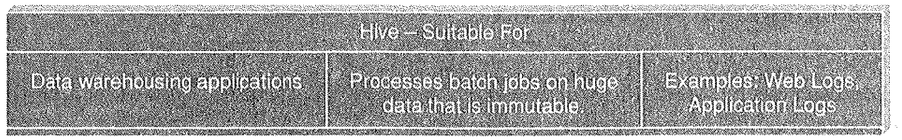


Figure 9.1 Hive – a data warehousing tool.

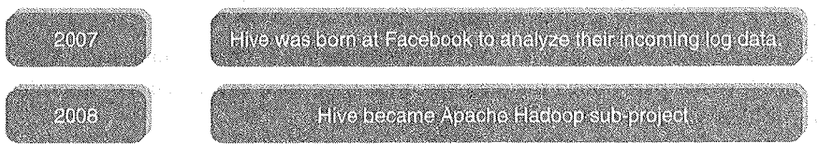


Figure 9.2 History of Hive.

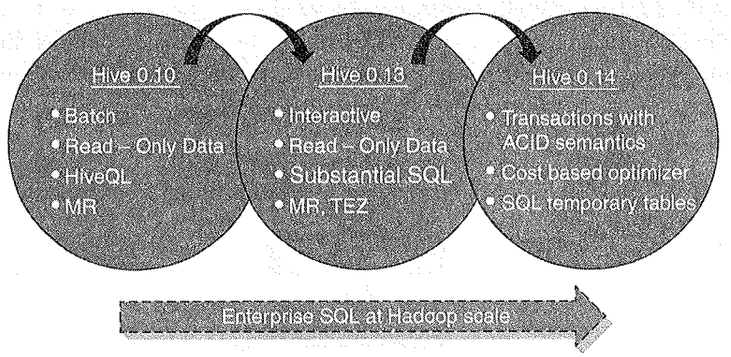


Figure 9.3 Recent releases of Hive.

Hive provides HQL (Hive Query Language) which is similar to SQL. Hive compiles SQL queries into MapReduce jobs and then runs the job in the Hadoop Cluster. Hive provides extensive data type functions and formats for data summarization and analysis.

9.1.1 History of Hive and Recent Releases of Hive

The history of Hive and recent releases of Hive are illustrated pictorially in Figures 9.2 and 9.3, respectively.

9.1.2 Hive Features

1. It is similar to SQL.
2. HQL is easy to code.
3. Hive supports rich data types such as structs, lists, and maps.
4. Hive supports SQL filters, group-by and order-by clauses.
5. Custom Types, Custom Functions can be defined.

9.1.3 Hive Integration and Work Flow

Figure 9.4 depicts the flow of log file analysis.

Hourly Log Data can be stored directly into HDFS and then data cleansing is performed on the log file. Finally Hive table(s) can be created to query the log file.

9.1.4 Hive Data Units

1. **Databases:** The namespace for tables.
2. **Tables:** Set of records that have similar schema.
3. **Partitions:** Logical separations of data based on classification of given information as per specific attributes. Once hive has partitioned the data based on a specified key, it starts to assemble the records into specific folders as and when the records are inserted.
4. **Buckets (or Clusters):** Similar to partitions but uses hash function to segregate data and determines the cluster or bucket into which the record should be placed.

Figure 9.5 shows how these data units are arranged in a Hive Cluster.
Figure 9.6 describes the semblance of Hive structure with database.

A database contains several tables. Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as a sub-directory. Bucketed tables are stored as a file.

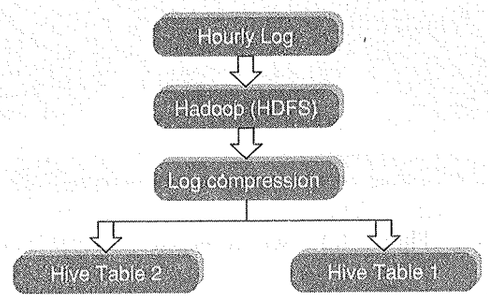


Figure 9.4 Flow of log analysis file.

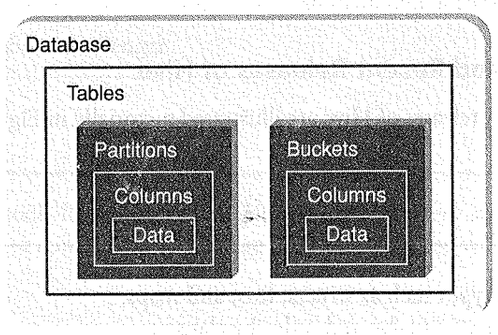


Figure 9.5 Data units as arranged in a Hive.

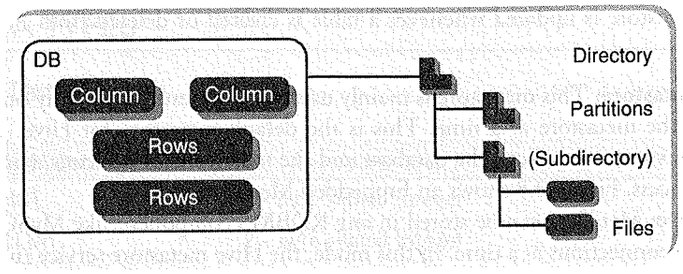


Figure 9.6 Semblance of Hive structure with database.

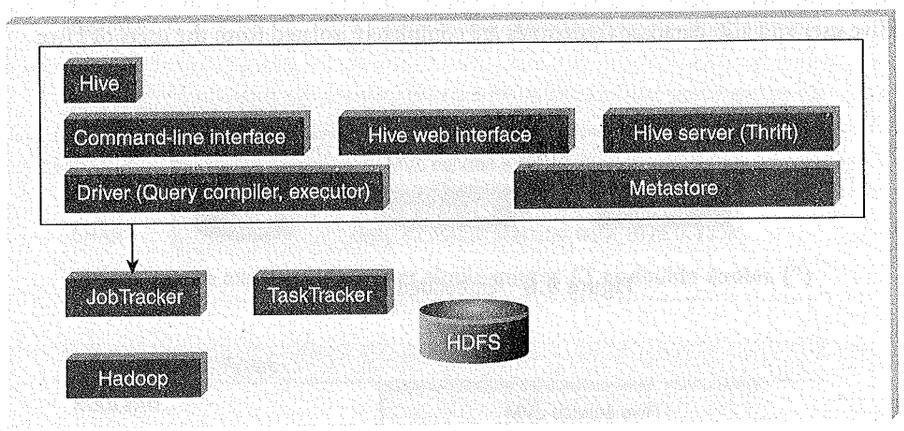


Figure 9.7 Hive architecture.

9.2 Hive Architecture

Hive Architecture is depicted in Figure 9.7. The various parts are as follows:

1. **Hive Command-Line Interface (Hive CLI):** The most commonly used interface to interact with Hive.
2. **Hive Web Interface:** It is a simple Graphic User Interface to interact with Hive and to execute query.
3. **Hive Server:** This is an optional server. This can be used to submit Hive Jobs from a remote client.
4. **JDBC / ODBC:** Jobs can be submitted from a JDBC Client. One can write a Java code to connect to Hive and submit jobs on it.
5. **Driver:** Hive queries are sent to the driver for compilation, optimization and execution.
6. **Metastore:** Hive table definitions and mappings to the data are stored in a Metastore. A Metastore consists of the following:
   - **Metastore service:** Offers interface to the Hive.
   - **Database:** Stores data definitions, mappings to the data and others.

The metadata which is stored in the metastore includes IDs of Database, IDs of Tables, IDs of Indexes, etc., the time of creation of a Table, the Input Format used for a Table, the Output Format used for

a Table, etc. The metastore is updated whenever a table is created or deleted from Hive. There are three kinds of metastore.

1. **Embedded Metastore:** This metastore is mainly used for unit tests. Here, only one process is allowed to connect to the metastore at a time. This is the default metastore for Hive. It is Apache Derby Database. In this metastore, both the database and the metastore service runs, embedded in the main Hive Server process. Figure 9.8 shows an Embedded Metastore.
2. **Local Metastore:** Metadata can be stored in any RDBMS component like MySQL. Local metastore allows multiple connections at a time. In this mode, the Hive metastore service runs in the main Hive Server process, but the metastore database runs in a separate process, and can be on a separate host. Figure 9.9 shows a Local Metastore.
3. **Remote Metastore:** In this, the Hive driver and the metastore interface run on different JVMs (which can run on different machines as well) as in Figure 9.10. This way the database can be fire-walled from the Hive user and also database credentials are completely isolated from the users of Hive.

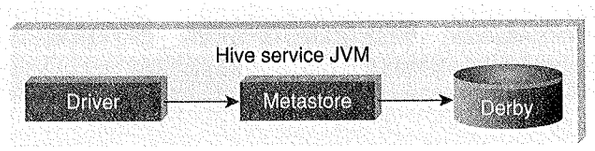


Figure 9.8 Embedded Metastore.

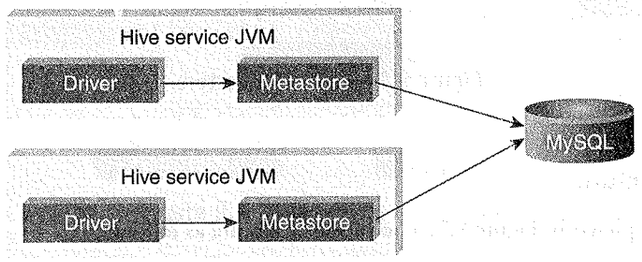


Figure 9.9 Local Metastore.

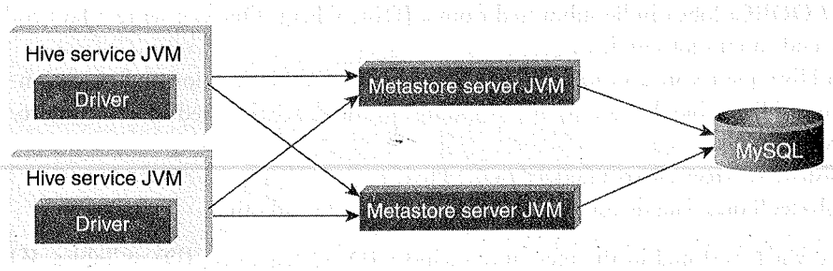


Figure 9.10 Remote Metastore.

9.3 HIVE DATA TYPES

9.3.1 Primitive Data Types

Numeric Data Type	
TINYINT	1 - byte signed integer
SMALLINT	2 - byte signed integer
INT	4 - byte signed integer
BIGINT	8 - byte signed integer
FLOAT	4 - byte single-precision floating-point
DOUBLE	8 - byte double-precision floating-point number

String Types	
STRING	
VARCHAR	Only available starting with Hive 0.12.0
CHAR	Only available starting with Hive 0.13.0
Strings can be expressed in either single quotes (') or double quotes (")	

Miscellaneous Types	
BOOLEAN	
BINARY	Only available starting with Hive

9.3.2 Collection Data Types

Collection Data Types	
STRUCT	Similar to **'C'** struct. Fields are accessed using dot notation. E.g.: struct('John', 'Doe')
MAP	A collection of key - value pairs. Fields are accessed using [] notation. E.g.: map('first', 'John', 'last', 'Doe')
ARRAY	Ordered sequence of same types. Fields are accessed using array index. E.g.: array('John', 'Doe')

9.4 HIVE FILE FORMAT

The file formats in Hive specify how records are encoded in a file.

9.4.1 Text File

The default file format is text file. In this format, each record is a line in the file. In text file, different control characters are used as delimiters. The delimiters are ^A (octal 001, separates all fields), ^B (octal 002,

separates the elements in the array or struct), ^C (octal 003, separates key–value pair), and \n. The term **field** is used when overriding the default delimiter. The supported text files are CSV and TSV. JSON or XML documents too can be specified as text file.

9.4.2 Sequential File

Sequential files are flat files that store binary key–value pairs. It includes compression support which reduces the CPU, I/O requirement.

9.4.3 RCFile (Record Columnar File)

RCFile stores the data in **Column Oriented Manner** which ensures that **Aggregation** operation is not an expensive operation. For example, consider a table which contains four columns as shown in Table 9.1.

Instead of only partitioning the table horizontally like the row-oriented DBMS (row-store), RCFile partitions this table first horizontally and then vertically to serialize the data. Based on the user-specified value, first the table is partitioned into multiple row groups horizontally. Depicted in Table 9.2, the table shown in Table 9.1 is partitioned into two row groups by considering three rows as the size of each row group.

Next, in every row group RCFile partitions the data vertically like column-store. So the table will be serialized as shown in Table 9.3.

Table 9.1 A table with four columns

C1	C2	C3	C4
11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44
51	52	53	54

Table 9.2 Table with two row groups

Row Group 1			
C1	C2	C3	C4
11	12	13	14
21	22	23	24
31	32	33	34

Row Group 2			
C1	C2	C3	C4
41	42	43	44
51	52	53	54

Table 9.3 Table in RCFile Format

Row Group 1	Row Group 2
11, 21, 31;	41, 51;
12, 22, 32;	42, 52;
13, 23, 33;	43, 53;
14, 24, 34;	44, 54;

9.5 HIVE QUERY LANGUAGE (HQL)

Hive query language provides basic SQL like operations. Here are few of the tasks which HQL can do easily.

1. Create and manage tables and partitions.
2. Support various Relational, Arithmetic, and Logical Operators.
3. Evaluate functions.
4. Download the contents of a table to a local directory or result of queries to HDFS directory.

9.5.1 DDL (Data Definition Language) Statements

These statements are used to build and modify the tables and other objects in the database. The DDL commands are as follows:

1. Create/Drop/Alter Database
2. Create/Drop/Truncate Table
3. Alter Table/Partition/Column
4. Create/Drop/Alter View
5. Create/Drop/Alter Index
6. Show
7. Describe

9.5.2 DML (Data Manipulation Language) Statements

These statements are used to retrieve, store, modify, delete, and update data in database. The DML commands are as follows:

1. Loading files into table.
2. Inserting data into Hive Tables from queries.

Note: Hive 0.14 supports update, delete, and transaction operations.

9.5.3 Starting Hive Shell

To start Hive, go to the installation path of Hive and type as below:

```
[root@volgalnx005 ~]# hive

Logging initialized using configuration in jar:file:/root/Desktop/VMDATA/Hive/hi
ve/lib/hive-common-0.14.0.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/root/Desktop/VMDATA/Hadoop/hadoop/share/hadoo
p/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/root/Desktop/VMDATA/Hive/hive/lib/hive-jdbc-0
.14.0-standalone.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
hive>
```

The sections have been designed as follows:

Objective:	What is it that we are trying to achieve here?
Input (optional):	What is the input that has been given to us to act upon?
Act:	The actual statement/command to accomplish the task at hand.
Outcome:	The result/output as a consequence of executing the statement.

9.5.4 Database

A database is like a container for data. It has a collection of tables which houses the data.

Objective: To create a database named "STUDENTS" with comments and database properties.

Act:

```
CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT 'STUDENT Details'
WITH DBPROPERTIES ('creator' = 'JOHN');
```

Outcome:

```
hive> CREATE DATABASE IF NOT EXISTS STUDENTS COMMENT 'STUDENT Details' WITH DBPROPERTIES ('creato
r' = 'JOHN');
OK
Time taken: 0.536 seconds
hive>
```

Objective: To display a list of all databases.

Act:

```
SHOW DATABASES;
```

Outcome:

```
hive> SHOW DATABASES;
OK
students
Time taken: 0.082 seconds, Fetched: 22 row(s)
hive>
```

Objective: To describe a database.

Act:

```
DESCRIBE DATABASE STUDENTS;
```

Note: Shows only DB name, comment, and DB directory.

Outcome:

```
hive> DESCRIBE DATABASE STUDENTS;
OK
students        STUDENT Details hdfs://volgalnx010.ad.infosys.com:9000/user/hive/warehouse/studen
ts.db   root    USER
Time taken: 0.03 seconds, Fetched: 1 row(s)
hive>
```

Objective: To describe the extended database.

Act:

```
DESCRIBE DATABASE EXTENDED STUDENTS;
```

Note: Shows DB properties also.

Outcome:

```
hive> DESCRIBE DATABASE EXTENDED STUDENTS;
OK
students        STUDENT Details hdfs://volgalnx010.ad.infosys.com:9000/user/hive/warehouse/studen
ts.db   root    USER    {creator=JOHN}
Time taken: 0.027 seconds, Fetched: 1 row(s)
hive>
```

Objective: To alter the database properties.

Act:

```
ALTER DATABASE STUDENTS SET DBPROPERTIES ('edited-by' = 'JAMES');
```

Note: In Hive, it is not possible to unset the DB properties.

Outcome:

```
hive> ALTER DATABASE STUDENTS SET DBPROPERTIES ('edited-by' = 'JAMES');
OK
Time taken: 0.086 seconds
hive>
```

Objective: To make the database as current working database.

Act:

```
USE STUDENTS;
```

Outcome:

```
hive> USE STUDENTS;
OK
Time taken: 0.02 seconds
hive>
```

Objective: To drop database.

Act:

```
DROP DATABASE STUDENTS;
```

Note: Hive creates database in the warehouse directory of Hive as shown below:

Contents of directory /user/hive/warehouse

Goto : /user/hive/warehouse go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
students.db	dir				2015-02-24 21:50	rwxr-xr-x	root	supergroup

9.5.5 Tables

Hive provides two kinds of table: Managed and External Table.

9.5.5.1 Managed Table

1. Hive stores the Managed tables under the warehouse folder under Hive.
2. The complete life cycle of table and data is managed by Hive.
3. When the internal table is dropped, it drops the data as well as the metadata.

Objective: To create managed table named 'STUDENT'.

Act:

```
CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW
FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT DELIMITED F
IELDS TERMINATED BY '\t';
OK
Time taken: 0.355 seconds
hive>
```

Objective: To describe the "STUDENT" table.

Act:

```
DESCRIBE STUDENT;
```

Outcome:

```
hive> DESCRIBE STUDENT;
OK
rollno                  int
name                    string
gpa                     float
Time taken: 0.163 seconds, Fetched: 3 row(s)
hive>
```

Note: Hive creates managed table in the warehouse directory of Hive as shown below:

Contents of directory /user/hive/warehouse/students.db

Goto : /user/hive/warehouse/student go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
student	dir				2015-02-24 22:03	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2015.

9.5.5.2 External or Self-Managed Table

1. When the table is dropped, it retains the data in the underlying location.
2. **External** keyword is used to create an external table.
3. **Location** needs to be specified to store the dataset in that particular location.

Objective: To create external table named 'EXT_STUDENT'.

Act:

```
CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name
STRING,gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
LOCATION '/STUDENT_INFO;
```

Outcome:

```
hive> CREATE EXTERNAL TABLE IF NOT EXISTS EXT_STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMA
T DELIMITED FIELDS TERMINATED BY '\t'  LOCATION '/STUDENT_INFO';
OK
Time taken: 0.123 seconds
hive>
```

Note: Hive creates the external table in the specified location.

9.5.5.3 Loading Data into Table from File

Objective: To load data into the table from file named student.tsv.

Act:

```
LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE
EXT_STUDENT;
```

Note: Local keyword is used to load the data from the local file system. To load the data from HDFS, remove local key word from the statement.

Outcome:

```
hive> LOAD DATA LOCAL INPATH  '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE EXT_STUDENT;
Loading data to table students.ext_student
Table students.ext_student stats: [numFiles=0, numRows=0, totalSize=0, rawDataSize=0]
OK
Time taken: 5.034 seconds
hive>
```

Hive loads the file in the specified location as shown below:

Contents of directory /STUDENT_INFO

Goto : /STUDENT_INFO go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
student.tsv	file	121 B	3	128 MB	2015-02-24 22:19	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2015.

File: /STUDENT_INFO/student.tsv

Goto /STUDENT_INFO go

Go back to dir listing
Advanced view/download options

```
1001    John    3.0
1002    Jack    4.0
1003    Smith   4.5
1004    Jamesh  4.2
1005    Joshi   3.5
1006    Alex    4.0
1007    David   4.2
1008    Scott   3.0
```

9.5.5.4 Collection Data Types

Objective: To work with collection data types.

Input:
1001,John,Smith:Jones,Mark1!45:Mark2!46:Mark3!43
1002,Jack,Smith:Jones,Mark1!46:Mark2!47:Mark3!42

Act:

```
CREATE TABLE STUDENT_INFO (rollno INT,name String, sub ARRAY<STRING>,marks
MAP<STRING,INT>)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
COLLECTION ITEMS TERMINATED BY ':'
MAP KEYS TERMINATED BY '!';
LOAD DATA LOCAL INPATH '/root/hivedemos/studentinfo.csv' INTO TABLE
STUDENT_INFO;
```

Outcome:

```
hive> CREATE TABLE STUDENT_INFO (rollno INT,name String, sub ARRAY<STRING>,marks MAP<STRING,FLOAT>)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > COLLECTION ITEMS TERMINATED BY ':'
    > MAP KEYS TERMINATED BY '!';
OK
Time taken: 0.112 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/studentinfo.csv' INTO TABLE STUDENT_INFO;
Loading data to table students.student_info
Table students.student_info stats: [numFiles=1, totalSize=109]
OK
Time taken: 0.397 seconds
hive>
```

9.5.5.5 Querying Table

Objective: To retrieve the student details from "EXT_STUDENT" table.

Act:

```
SELECT * from EXT_STUDENT;
```


Outcome:

```
hive> select * from EXT_STUDENT;
OK
1001    John    3.0
1002    Jack    4.0
1003    Smith   4.5
1004    Scott   4.2
1005    Joshi   3.5
1006    Alex    4.5
1007    David   4.2
1008    James   4.0
1009    John    3.0
1010    Joshi   3.5
Time taken: 0.054 seconds, Fetched: 10 row(s)
hive>
```

Objective: Querying Collection Data Types.

Act:

```
SELECT * from STUDENT_INFO;
SELECT NAME,SUB FROM STUDENT_INFO;
// To retrieve value of Mark1
SELECT NAME, MARKS['Mark1'] from STUDENT_INFO;
// To retrieve subordinate (array) value
SELECT NAME,SUB[0] FROM STUDENT_INFO;
```

Outcome:

```
hive> SELECT * from STUDENT_INFO;
OK
1001    John    ["Smith","Jones"]       {"Mark1":45,"Mark2":46,"Mark3":43}
1002    Jack    ["Smith","Jones"]       {"Mark1":46,"Mark2":47,"Mark3":42}
Time taken: 0.044 seconds, Fetched: 2 row(s)
```

```
hive> SELECT NAME,SUB FROM STUDENT_INFO;
OK
John    ["Smith","Jones"]
Jack    ["Smith","Jones"]
Time taken: 0.061 seconds, Fetched: 2 row(s)
hive>
```

```
hive> SELECT NAME, MARKS['Mark1']  from STUDENT_INFO;
OK
John    45
Jack    46
Time taken: 0.06 seconds, Fetched: 2 row(s)
hive>
```

```
hive> SELECT NAME,SUB[0] FROM STUDENT_INFO;
OK
John    Smith
Jack    Smith
Time taken: 0.071 seconds, Fetched: 2 row(s)
hive>
```

9.5.6 Partitions

In Hive, the query reads the entire dataset even though a where clause filter is specified on a particular column. This becomes a bottleneck in most of the MapReduce jobs as it involves huge degree of I/O. So it is necessary to reduce I/O required by the MapReduce job to improve the performance of the query. A very common method to reduce I/O is data partitioning.

Partitions split the larger dataset into more meaningful chunks.

Hive provides two kinds of partitions: Static Partition and Dynamic Partition.

9.5.6.1 Static Partition

Static partitions comprise columns whose values are known at compile time.

Objective: To create static partition based on "gpa" column.

Act:

```
CREATE TABLE IF NOT EXISTS STATIC_PART_STUDENT (rollno INT, name STRING)
PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED
BY '\t';
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STATIC_PART_STUDENT(rollno INT,name STRING) PARTITIONED BY (gpa FLOAT) RO
W FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.105 seconds
hive>
```

Objective: Load data into partition table from table.

Act:

```
INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0)
SELECT rollno, name from EXT_STUDENT where gpa=4.0;
```

Outcome:

```
hive> INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0) SELECT rollno,name from EXT_STUDENT
where gpa=4.0;
Query ID = root_20150224230404_4500d58a-cb21-4912-ba40-788e5cf8f9da
Total jobs = 3
```

Hive creates the folder for the value specified in the partition.

Contents of directory /user/hive/warehouse/students.db

Goto : /user/hive/warehouse/student go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
static_part_student	dir				2015-02-24 23:04	rwxr-xr-x	root	supergroup
student	dir				2015-02-24 22:03	rwxr-xr-x	root	supergroup
student_info	dir				2015-02-24 22:54	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Log directory

Contents of directory /user/hive/warehouse/students.db/static_part_student

Goto : /user/hive/warehouse/student go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
gpa=4.0	dir				2015-02-24 23:04	rwxr-xr-x	root	supergroup

Go back to DFS home

Local logs

Log directory

File: /user/hive/warehouse/students.db/static_part_student/gpa=4.0/000000_0

Goto : /user/hive/warehouse/student go

Go back to dir listing
Advanced view/download options

```
1002    Jack
1008    James
```

Objective: To add one more static partition based on "gpa" column using the "alter" statement.

Act:

```
ALTER TABLE STATIC_PART_STUDENT ADD PARTITION (gpa=3.5);
INSERT OVERWRITE TABLE STATIC_PART_STUDENT PARTITION (gpa =4.0) SELECT
rollno,name from EXT_STUDENT where gpa=4.0;
```

Outcome:

```
hive> ALTER TABLE STATIC_PART_STUDENT ADD PARTITION (gpa=3.5);
OK
Time taken: 0.166 seconds
hive>
```

Contents of directory /user/hive/warehouse/students.db/static_part_student

Goto : /user/hive/warehouse/student go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
gpa=3.5	dir				2015-02-24 23:09	rwxr-xr-x	root	supergroup
gpa=4.0	dir				2015-02-24 23:11	rwxr-xr-x	root	supergroup

Go back to DFS home

9.5.6.2 Dynamic Partition

Dynamic partition have columns whose values are known only at Execution Time.

Objective: To create dynamic partition on column date.

Act:

```
CREATE TABLE IF NOT EXISTS DYNAMIC_PART_STUDENT(rollno INT,name STRING)
PARTITIONED BY (gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED
BY '\t';
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS DYNAMIC_PART_STUDENT(rollno INT,name STRING) PARTITIONED BY (gpa FLOAT) R
OW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.166 seconds
hive>
```

Objective: To load data into a dynamic partition table from table.

Act:

SET hive.exec.dynamic.partition = true;

SET hive.exec.dynamic.partition.mode = nonstrict;

Note: The dynamic partition strict mode requires at least one static partition column. To turn this off, set hive.exec.dynamic.partition.mode=nonstrict

INSERT OVERWRITE TABLE DYNAMIC_PART_STUDENT PARTITION (gpa) SELECT rollno,name,gpa from EXT_STUDENT;

Outcome:

Contents of directory /user/hive/warehouse/students.db/dynamic_part_student

Goto : /user/hive/warehouse/student go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
gpa=3.0	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=3.5	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=4.0	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=4.2	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup
gpa=4.5	dir				2015-02-24 23:16	rwxr-xr-x	root	supergroup

Go back to DFS home

Note: Create partition for all values.

9.5.7 Bucketing

Bucketing is similar to partition. However, there is a subtle difference between partition and bucketing. In a partition, you need to create partition for each unique value of the column. This may lead to situations where you may end up with thousands of partitions. This can be avoided by using Bucketing in which you can limit the number of buckets to create. A bucket is a file whereas a partition is a directory.

Objective: To learn about bucket in hive.

Act:

CREATE TABLE IF NOT EXISTS STUDENT (rollno INT,name STRING,grade FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' INTO TABLE STUDENT;

Set below property to enable bucketing.

set hive.enforce.bucketing=true;
// To create a bucketed table having 3 buckets
CREATE TABLE IF NOT EXISTS STUDENT_BUCKET (rollno INT,name STRING,grade FLOAT)
CLUSTERED BY (grade) into 3 buckets;

// Load data to bucketed table
FROM STUDENT
INSERT OVERWRITE TABLE STUDENT_BUCKET
SELECT rollno,name,grade;
// To display content of first bucket
SELECT DISTINCT GRADE FROM STUDENT_BUCKET
TABLESAMPLE(BUCKET 1 OUT OF 3 ON GRADE);

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STUDENT (rollno INT,name STRING,grade FLOAT)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.101 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' INTO TABLE STUDENT;
Loading data to table book.student
Table book.student stats: [numFiles=1, totalSize=145]
OK
Time taken: 0.536 seconds
hive>
```

```
hive> set hive.enforce.bucketing=true;
hive>
```

```
hive> CREATE TABLE IF NOT EXISTS STUDENT_BUCKET (rollno INT,name STRING,grade FLOAT)
    > CLUSTERED BY (grade) into 3 buckets;
OK
Time taken: 0.101 seconds
hive>
```

```
hive> FROM STUDENT
    > INSERT OVERWRITE TABLE STUDENT_BUCKET
    > SELECT rollno,name,grade;
```

3 buckets have been created as shown below:

Contents of directory /user/hive/warehouse/book.db/student_bucket

Goto : /user/hive/warehouse/book.db go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
000000_0	file	59 B	3	128 MB	2015-03-10 22:29	rw-r--r--	root	supergroup
000001_0	file	59 B	3	128 MB	2015-03-10 22:29	rw-r--r--	root	supergroup
000002_0	file	28 B	3	128 MB	2015-03-10 22:29	rw-r--r--	root	supergroup

Go back to DFS home

Local logs

Log directory

Hadoop, 2015.

```
hive>
    > SELECT DISTINCT GRADE FROM STUDENT_BUCKET
    > TABLESAMPLE(BUCKET 1 OUT OF 3 ON GRADE);
OK
4.0
4.2
Time taken: 21.117 seconds, Fetched: 2 row(s)
hive>
```

9.5.8 Views

In Hive, view support is available only in version starting from 0.6. Views are purely logical object.

Objective: To create a view table named "STUDENT_VIEW".

Act:

```
CREATE VIEW STUDENT_VIEW AS SELECT rollno, name FROM EXT_STUDENT;
```

Outcome:

```
hive> CREATE VIEW STUDENT_VIEW AS SELECT rollno,name FROM EXT_STUDENT;
OK
Time taken: 0.606 seconds
hive>
```

Objective: Querying the view "STUDENT_VIEW".

Act:

```
SELECT * FROM STUDENT_VIEW LIMIT 4;
```

Outcome:

```
hive> SELECT * FROM STUDENT_VIEW LIMIT 4;
OK
1001    John
1002    Jack
1003    Smith
1004    Scott
Time taken: 0.279 seconds, Fetched: 4 row(s)
hive>
```

Objective: To drop the view "STUDENT_VIEW".

Act:

```
DROP VIEW STUDENT_VIEW;
```

Outcome:

```
hive> DROP VIEW STUDENT_VIEW;
OK
Time taken: 0.452 seconds
hive>
```

9.5.9 Sub-Query

In Hive, sub-queries are supported only in the FROM clause (Hive 0.12). You need to specify name for sub-query because every table in a FROM clause has a name. The columns in the sub-query select list should have unique names. The columns in the subquery select list are available to the outer query just like columns of a table.

Objective: Write a sub-query to count occurrence of similar words in the file.

Act:

```
CREATE TABLE docs (line STRING);
LOAD DATA LOCAL INPATH '/root/hivedemos/lines.txt' OVERWRITE INTO TABLE docs;
CREATE TABLE word_count AS
SELECT word, count(1) AS count FROM
(SELECT explode (split (line, ' ')) AS word FROM docs) w
GROUP BY word
ORDER BY word;
SELECT * FROM word_count;
```

Outcome:

```
hive> CREATE TABLE docs (line STRING);
OK
Time taken: 0.118 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/lines.txt' OVERWRITE INTO TABLE docs;
Loading data to table students.docs
Table students.docs stats: [numFiles=1, numRows=0, totalSize=91, rawDataSize=0]
OK
Time taken: 2.697 seconds
hive>
```

```
hive> CREATE TABLE word_count AS
    > SELECT word, count(1) AS count FROM
    > (SELECT explode (split (line, ' ')) AS word FROM docs) w
    > GROUP BY word
    > ORDER BY word;
```

```
hive> SELECT * FROM word_count;
OK
Hadoop  2
Hive    2
Introducing     1
Introduction    1
Pig     1
Session 3
Welcome 1
to      2
Time taken: 0.062 seconds, Fetched: 8 row(s)
hive>
```

Note: The explode() function takes an array as input and outputs the elements of the array as separate rows.

In Hive 0.13, sub-queries are supported in the where clause as well.

9.5.10 Joins

Joins in Hive is similar to the SQL Join.

Objective: To create JOIN between Student and Department tables where we use RollNo from both the tables as the join key.

Act:

```
CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE STUDENT;
CREATE TABLE IF NOT EXISTS DEPARTMENT(rollno INT,deptno int,name STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
LOAD DATA LOCAL INPATH '/root/hivedemos/department.tsv' OVERWRITE INTO TABLE DEPARTMENT;
SELECT a.rollno, a.name, a.gpa, b.deptno FROM STUDENT a JOIN DEPARTMENT b ON a.rollno = b.rollno;
```

Outcome:

```
hive> CREATE TABLE IF NOT EXISTS STUDENT(rollno INT,name STRING,gpa FLOAT) ROW FORMAT D
ELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.115 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/student.tsv' OVERWRITE INTO TABLE STUDENT
;
Loading data to table students.student
Table students.student stats: [numFiles=1, numRows=0, totalSize=145, rawDataSize=0]
OK
Time taken: 0.723 seconds
hive>
```

```
hive> CREATE TABLE IF NOT EXISTS DEPARTMENT(rollno INT,deptno int,name STRING) ROW FORM
AT DELIMITED FIELDS TERMINATED BY '\t';
OK
Time taken: 0.099 seconds
hive>
```

```
hive> LOAD DATA LOCAL INPATH '/root/hivedemos/department.tsv' OVERWRITE INTO TABLE DEPA
RTMENT;
Loading data to table students.department
Table students.department stats: [numFiles=1, numRows=0, totalSize=120, rawDataSize=0]
OK
Time taken: 0.442 seconds
hive>
```

```
hive> SELECT a.rollno, a.name, a.gpa, b.deptno  FROM STUDENT  a JOIN DEPARTMENT b ON a.
rollno = b.rollno;

OK
1001    John    3.0     101
1002    Jack    4.0     102
1003    Smith   4.5     103
1004    Scott   4.2     104
1005    Joshi   3.5     105
1006    Alex    4.5     101
1007    David   4.2     104
1008    James   4.0     102
Time taken: 115.282 seconds, Fetched: 8 row(s)
hive>
```

9.5.11 Aggregation

Hive supports aggregation functions like avg, count, etc.

Objective: To write the average and count aggregation function.

Act:

```
SELECT avg(gpa) FROM STUDENT;
SELECT count(*) FROM STUDENT;
```

Outcome:

```
hive> SELECT avg(gpa) FROM STUDENT;

OK
38.39999961853027
Time taken: 25.41 seconds, Fetched: 1 row(s)
hive>
```

```
hive> SELECT avg(gpa) FROM STUDENT;

OK
10
Time taken: 26.218 seconds, Fetched: 1 row(s)
hive>
```

9.5.12 Group By and Having

Data in a column or columns can be grouped on the basis of values contained therein by using "Group By". "Having" clause is used to filter out groups NOT meeting the specified condition.

Objective: To write group by and having function.

Act:

```
SELECT rollno, name,gpa FROM STUDENT GROUP BY rollno,name,gpa HAVING gpa > 4.0;
```

Outcome:

```
1003    Smith   4.5
1004    Scott   4.2
1006    Alex    4.5
1007    David   4.2
Time taken: 78.972 seconds, Fetched: 4 row(s)
hive>
```

9.6 RCFILE IMPLEMENTATION

RCFile (Record Columnar File) is a data placement structure that determines how to store relational tables on computer clusters.

Objective: To work with RCFILE Format.

Act:

```
CREATE TABLE STUDENT_RC( rollno int, name string,gpa float ) STORED AS RCFILE;
INSERT OVERWRITE table STUDENT_RC SELECT * FROM STUDENT;
SELECT SUM(gpa) FROM STUDENT_RC;
```

Outcome:

```
hive> CREATE TABLE STUDENT_RC( rollno int, name string,gpa float ) STORED AS RCFILE;
OK
Time taken: 0.093 seconds
hive>

hive> INSERT OVERWRITE table STUDENT_RC SELECT * from STUDENT;

hive> SELECT SUM(gpa) from STUDENT_RC;

OK
38.399999618530027
Time taken: 25.41 seconds, Fetched: 1 row(s)
hive>
```

Note: Stores the data in column oriented manner.

```
File: /user/hive/warehouse/students.db/student_rc/000000_0

Goto : /user/hive/warehouse/student  go

Go back to dir listing
Advanced view/download options

RCFhive.io.rcfile.column.number3V@@@@@@3@@%@@@
@..    @@@((@@@@@@@@@@@@@@JohnJackSmithScottJoshiAlexDavidJamesJohnJoshi@@@@@@$@@ff@`@@3@ff@@@@`
```

9.7 SERDE

SerDe stands for Serializer/Deserializer.

1. Contains the logic to convert unstructured data into records.
2. Implemented using Java.
3. Serializers are used at the time of writing.
4. Deserializers are used at query time (SELECT Statement).

Deserializer interface takes a binary representation or string of a record, converts it into a java object that Hive can then manipulate. Serializer takes a java object that Hive has been working with and translates it into something that Hive can write to HDFS.

Objective: To manipulate the XML data.

Input:

```
<employee> <empid>1001</empid> <name>John</name> <designation>Team Lead</designation>
</employee>
<employee> <empid>1002</empid> <name>Smith</name> <designation>Analyst</designation>
</employee>
```


Act:

```
CREATE TABLE XMLSAMPLE(xmldata string);
LOAD DATA LOCAL INPATH '/root/hivedemos/input.xml' INTO TABLE XMLSAMPLE;

CREATE TABLE xpath_table AS
SELECT xpath_int(xmldata,'employee/empid'),
xpath_string(xmldata,'employee/name'),
xpath_string(xmldata,'employee/designation')
FROM xmlsample;

SELECT * FROM xpath_table;
```

Outcome:

```
hive> CREATE TABLE XMLSAMPLE(xmldata string);
OK
Time taken: 0.244 seconds
hive>

hive> LOAD DATA LOCAL INPATH '/root/hivedemos/input.xml' INTO TABLE XMLSAMPLE;
Loading data to table students.xmlsample
Table students.xmlsample stats: [numFiles=1, totalSize=194]
OK
Time taken: 0.889 seconds
hive>

hive> CREATE TABLE xpath_table AS
    > SELECT xpath_int(xmldata,'employee/empid'),
    > xpath_string(xmldata,'employee/name'),
    > xpath_string(xmldata,'employee/designation')
    > FROM xmlsample;

hive> SELECT * FROM xpath_table;
OK
1001    John    Team Lead
1002    Smith   Analyst
Time taken: 0.064 seconds, Fetched: 2 row(s)
hive>
```

9.8 USER-DEFINED FUNCTION (UDF)

In Hive, you can use custom functions by defining the User-Defined Function (UDF).

Objective: Write a Hive function to convert the values of a field to uppercase.

Act:

```
package com.example.hive.udf;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
@Description(
 name="SimpleUDFExample")
```

```
public final class MyLowerCase extends UDF {
  public String evaluate(final String word) {
    return word.toLowerCase();
  }
}
```

Note: Convert this Java Program into Jar.

```
ADD JAR /root/hivedemos/UpperCase.jar;
CREATE TEMPORARY FUNCTION touppercase AS 'com.example.hive.udf.MyUpperCase';
SELECT TOUPPERCASE(name) FROM STUDENT;
```

Outcome:

```
hive> ADD JAR /root/hivedemos/UpperCase.jar;
Added [/root/hivedemos/UpperCase.jar] to class path
Added resources: [/root/hivedemos/UpperCase.jar]
hive> CREATE TEMPORARY FUNCTION touppercase AS 'com.example.hive.udf.MyUpperCase';
OK
Time taken: 0.014 seconds
hive>
```

```
hive> Select touppercase (name) from STUDENT;
OK
JOHN
JACK
SMITH
SCOTT
JOSHI
ALEX
DAVID
JAMES
JOHN
JOSHI
Time taken: 0.061 seconds, Fetched: 10 row(s)
hive>
```

REMIND ME

- Hive is a Data Warehousing tool.
- Hive is used to query structured data built on top of Hadoop.
- Hive provides HQL (Hive Query Language) which is similar to SQL.
- A Hive database contains several tables. Each table is constituted of rows and columns. In Hive, tables are stored as a folder and partition tables are stored as a sub-directory.
- Bucketed tables are stored as a file.

POINT ME (BOOKS)

- Programming Hive, Jason Rutherglen, O'Reilly Publication.

CONNECT ME (INTERNET RESOURCES)

- **http://en.wikipedia.org/wiki/RCFile**
- **https://cwiki.apache.org/confluence/display/Hive/DynamicPartitions**
- **https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL**
- **https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML**
- **Wrox Certified BigData Developer.**

TEST ME

A. Match Me

Column A	Column B
HQL	Web Logs
Database	struct, map
Complex Data Types	Set of records
Hive Application	Hive Query Language
Table	Namespace

Answers:

Column A	Column B
HQL	Hive Query Language
Database	Namespace
Complex Data Types	struct, map
Hive Application	Web Logs
Table	Set of records

B. Fill Me

1. The metastore consists of _____ and a _____.
2. The most commonly used interface to interact with Hive is _____.
3. The default metastore for Hive is _____.
4. Metastore contains _____ of Hive tables.
5. _____ is responsible for compilation, optimization, and execution of Hive queries.

Answers:

1. Metaservices, database
2. Command Line Interface
3. Derby
4. System Catalog
5. Driver

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1: PARTITION

Objective: To learn about partitions in hive.
Problem Description:
Create a partition table for customer schema to reward the customers based on their life time values.
Input:

Customer ID	Customers	Life Time Value
1001	Jack	25000
1002	Smith	8000
1003	David	12000
1004	John	15000
1005	Scott	12000
1006	Joshi	28000
1007	Ajay	12000
1008	Vinay	30000
1009	Joseph	21000

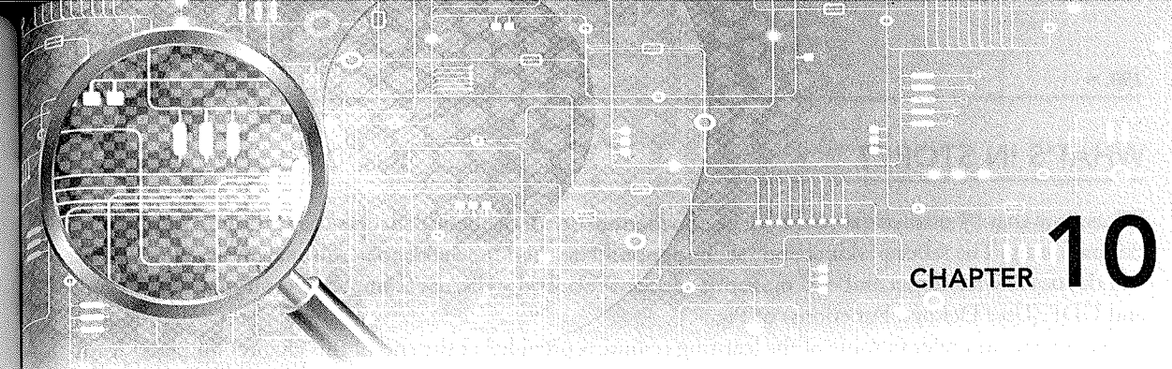
- Create a partition table if life time value is 12000.
- Create a partition table for all life time values.

ASSIGNMENT 2: HIVEQL

Objective: To learn about HiveQL statement.
Problem Description:
Create a data file for below schemas:

- **Order:** CustomerId, ItemId, ItemName, OrderDate, DeliveryDate
- **Customer:** CustomerId, CustomerName, Address, City, State, Country

1. Create a table for Order and Customer Data.
2. Write a HiveQL to find number of items bought by each customer.



CHAPTER 10

Introduction to Pig

BRIEF CONTENTS

"If you can't explain it simply, you don't understand it well enough."

– Albert Einstein, Physicist

WHAT'S IN STORE?

We assume that by now you would have become familiar with the basic concepts of HDFS and MapReduce Programming. The focus of this chapter will be to build on this knowledge to perform analysis using Pig. We will discuss few relational and eval operators of Pig. We will also discuss Complex Data Types, Piggy Bank, and UDF (User Defined Functions) of Pig.

We suggest you refer to some of the learning resources provided at the end of this chapter for better learning. We also suggest you to practice "Test Me" exercises.

10.1 WHAT IS PIG?

Apache Pig is a platform for data analysis. It is an alternative to MapReduce Programming. Pig was developed as a research project at Yahoo.

10.1.1 Key Features of Pig

1. It provides an **engine** for executing **data flows** (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
2. It provides a language called "**Pig Latin**" to express data flows.
3. Pig Latin contains operators for many of the traditional data operations such as join, filter, sort, etc.
4. It allows users to develop their own functions (User Defined Functions) for reading, processing, and writing data.

10.2 THE ANATOMY OF PIG

The main components of Pig are as follows:

1. Data flow language (**Pig Latin**).
2. Interactive shell where you can type Pig Latin statements (**Grunt**).
3. Pig interpreter and execution engine.

Refer Figure 10.1.

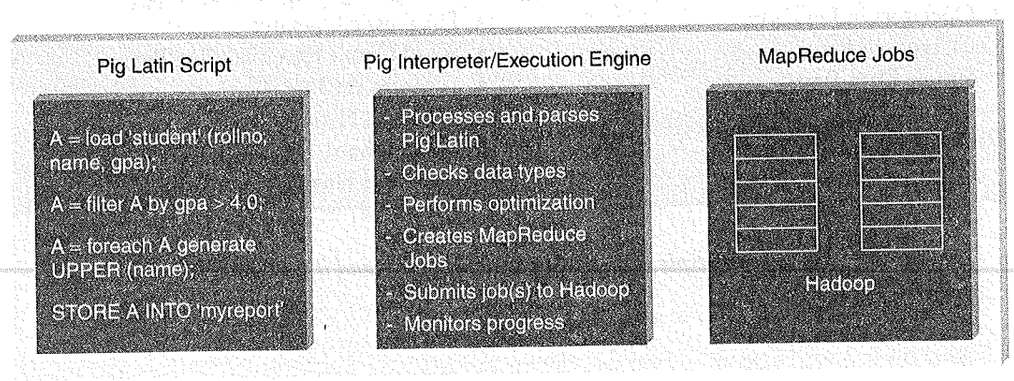


Figure 10.1 The anatomy of Pig.

10.3 PIG ON HADOOP

Pig runs on Hadoop. Pig uses both Hadoop Distributed File System and MapReduce Programming. By default, Pig reads input files from HDFS. Pig stores the intermediate data (data produced by MapReduce jobs) and the output in HDFS. However, Pig can also read input from and place output to other sources.

Pig supports the following:

1. HDFS commands.
2. UNIX shell commands.
3. Relational operators.
4. Positional parameters.
5. Common mathematical functions.
6. Custom functions.
7. Complex data structures.

10.4 PIG PHILOSOPHY

Figure 10.2 describes the Pig philosophy.

1. **Pigs Eat Anything:** Pig can process different kinds of data such as structured and unstructured data.
2. **Pigs Live Anywhere:** Pig not only processes files in HDFS, it also processes files in other sources such as files in the local file system.
3. **Pigs are Domestic Animals:** Pig allows you to develop user-defined functions and the same can be included in the script for complex operations.
4. **Pigs Fly:** Pig processes data quickly.

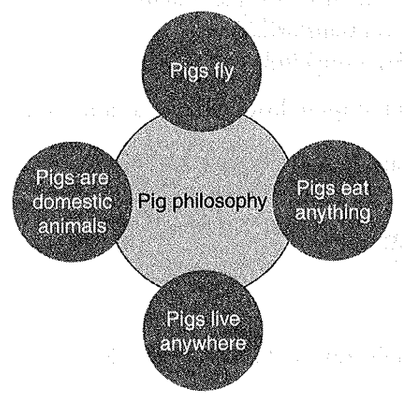


Figure 10.2 Pig philosophy.

10.5 USE CASE FOR PIG: ETL PROCESSING

Pig is widely used for "ETL" (**E**xtract, **T**ransform, and **L**oad). Pig can extract data from different sources such as ERP, Accounting, Flat Files, etc. Pig then makes use of various operators to perform transformation on the data and subsequently loads it into the data warehouse. Refer Figure 10.3.

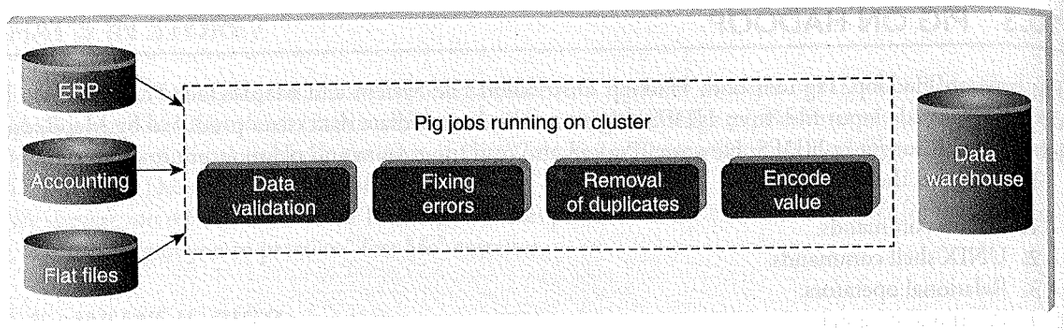


Figure 10.3 Pig: ETL Processing.

10.6 PIG LATIN OVERVIEW

10.6.1 Pig Latin Statements

1. Pig Latin statements are basic constructs to process data using Pig.
2. Pig Latin statement is an operator.
3. An operator in Pig Latin takes a relation as input and yields another relation as output.
4. Pig Latin statements include schemas and expressions to process data.
5. Pig Latin statements should end with a semi-colon.

Pig Latin Statements are generally ordered as follows:

1. **LOAD** statement that reads data from the file system.
2. Series of statements to perform transformations.
3. **DUMP** or **STORE** to display/store result.

The following is a simple Pig Latin script to load, filter, and store "student" data.

```
A = load 'student' (rollno, name, gpa);
A = filter A by gpa > 4.0;
A = foreach A generate UPPER (name);
STORE A INTO 'myreport'
```

Note: In the above example **A** is a relation and NOT a variable.

10.6.2 Pig Latin: Keywords

Keywords are reserved. It cannot be used to name things.

10.6.3 Pig Latin: Identifiers

1. Identifiers are names assigned to fields or other data structures.
2. It should begin with a letter and should be followed only by letters, numbers, and underscores.

Table 10.1 Valid and invalid identifiers

Valid Identifier	**Y**	**A1**	**A1_2014**	**Sample**
Invalid Identifier	**5**	**Sales$**	**Sales%**	**_Sales**

Table 10.1 describes valid and invalid identifiers.

10.6.4 Pig Latin: Comments

In Pig Latin two types of comments are supported:

1. Single line comments that begin with "--".
2. Multiline comments that begin with "/* and end with */".

10.6.5 Pig Latin: Case Sensitivity

1. Keywords are *not* case sensitive such as LOAD, STORE, GROUP, FOREACH, DUMP, etc.
2. Relations and paths are case-sensitive.
3. Function names are case sensitive such as PigStorage, COUNT.

10.6.6 Operators in Pig Latin

Table 10.2 describes operators in Pig Latin.

Table 10.2 Operators in Pig Latin

Arithmetic	Comparison	Null	Boolean
+	==	IS NULL	AND
–	!=	IS NOT NULL	OR
*	<		NOT
/	>		
%	<=		
	>=		

10.7 DATA TYPES IN PIG

10.7.1 Simple Data Types

Table 10.3 describes simple data types supported in Pig. In Pig, fields of unspecified types are considered as an array of bytes which is known as bytearray.

Null: In Pig Latin, **NULL** denotes a value that is unknown or is non-existent.

10.7.2 Complex Data Types

Table 10.4 describes complex data types in Pig.

Table 10.3 Simple data types supported in Pig

Name	Description
Int	Whole numbers
Long	Large whole numbers
Float	Decimals
Double	Very precise decimals
Chararray	Text strings
Bytearray	Raw bytes
Datetime	Datetime
Boolean	true or false

Table 10.4 Complex data types in Pig

Name	Description
Tuple	An ordered set of fields. **Example: (2,3)**
Bag	A collection of tuples. **Example: {(2,3),(7,5)}**
map	key, value pair **(open # Apache)**

10.8 RUNNING PIG

You can run Pig in two ways:

1. Interactive Mode.
2. Batch Mode.

10.8.1 Interactive Mode

You can run Pig in interactive mode by invoking **grunt** shell. Type pig to get grunt shell as shown below.

```
root@volgalnx010:~
[root@volgalnx010 ~]# pig
2015-02-23 21:07:38,916 [main] INFO  org.apache.pig.Main - Apache Pig version 0.12.0-cdh5.
1.3 (rexported) compiled Sep 16 2014, 20:39:43
2015-02-23 21:07:38,917 [main] INFO  org.apache.pig.Main - Logging error messages to: /roo
t/pig_1424705858915.log
2015-02-23 21:07:38,934 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file
/root/.pigbootup not found
2015-02-23 21:07:39,313 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - ma
pred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2015-02-23 21:07:39,313 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs
.default.name is deprecated. Instead, use fs.defaultFS
2015-02-23 21:07:39,313 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecut
ionEngine - Connecting to hadoop file system at: hdfs://volgalnx010.ad.infosys.com:9000
2015-02-23 21:07:39,800 [main] WARN  org.apache.hadoop.util.NativeCodeLoader - Unable to l
oad native-hadoop library for your platform... using builtin-java classes where applicable
2015-02-23 21:07:40,234 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs
.default.name is deprecated. Instead, use fs.defaultFS
grunt>
```


Once you get the grunt prompt, you can type the Pig Latin statement as shown below.

```
grunt> A = load '/pigdemo/student.tsv' as (rollno, name, gpa);
grunt> DUMP A;
```

Here, the path refers to HDFS path and DUMP displays the result on the console as shown below.

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
grunt>
```

10.8.2 Batch Mode

You need to create **"Pig Script"** to run pig in batch mode. Write Pig Latin statements in a file and save it with **.pig** extension.

10.9 EXECUTION MODES OF PIG

You can execute pig in two modes:

1. Local Mode.
2. MapReduce Mode.

10.9.1 Local Mode

To run pig in local mode, you need to have your files in the local file system.

Syntax:

```
pig –x local filename
```

10.9.2 MapReduce Mode

To run pig in MapReduce mode, you need to have access to a Hadoop Cluster to read /write file. This is the default mode of Pig.

Syntax:

```
pig filename
```

10.10 HDFS COMMANDS

You can work with all HDFS commands in Grunt shell. For example, you can create a directory as shown below.

```
grunt> fs -mkdir /piglatindemos;
grunt>
```

The sections have been designed as follows:

Objective: What is it that we are trying to achieve here?
Input: What is the input that has been given to us to act upon?
Act: The actual statement/command to accomplish the task at hand.
Outcome: The result/output as a consequence of executing the statement.

10.11 RELATIONAL OPERATORS

10.11.1 FILTER

FILTER operator is used to select tuples from a relation based on specified conditions.

Objective: Find the tuples of those student where the GPA is greater than 4.0.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = filter A by gpa > 4.0;
DUMP B;
```

Output:

```
(1003,Smith,4.5)
(1004,Scott,4.2)
[root@volgalnx010 pigdemos]#
```

10.11.2 FOREACH

Use ***FOREACH*** when you want to do data transformation based on columns of data.

Objective: Display the name of all students in uppercase.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = foreach A generate UPPER (name);
DUMP B;
```

Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(JOSHI)
[root@volgalnx010 pigdemos]#
```


10.11.3 GROUP

GROUP operator is used to group data.

Objective: Group tuples of students based on their GPA.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = GROUP A BY gpa;
DUMP B;
```

Output:

```
(3.0,{(1001,John,3.0),(1001,John,3.0)})
(3.5,{(1005,Joshi,3.5),(1005,Joshi,3.5)})
(4.0,{(1008,James,4.0),(1002,Jack,4.0)})
(4.2,{(1007,David,4.2),(1004,Scott,4.2)})
(4.5,{(1006,Alex,4.5),(1003,Smith,4.5)})
[root@volgalnx010 pigdemos]#
```

10.11.4 DISTINCT

DISTINCT operator is used to remove duplicate tuples. In Pig, DISTINCT operator works on the entire tuple and NOT on individual fields.

Objective: To remove duplicate tuples of students.
Input:
Student (rollno:int,name:chararray,gpa:float)
Input:

1001	John	3.0
1002	Jack	4.0
1003	Smith	4.5
1004	Scott	4.2
1005	Joshi	3.5
1006	Alex	4.5
1007	David	4.2
1008	James	4.0
1001	John	3.0
1005	Joshi	3.5

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = DISTINCT A;
DUMP B;
```

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
(1006,Alex,4.5)
(1007,David,4.2)
(1008,James,4.0)
[root@volgalnx010 pigdemos]#
```

10.11.5 LIMIT

LIMIT operator is used to limit the number of output tuples.

Objective: Display the first 3 tuples from the "student" relation.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = LIMIT A 3;
DUMP B;
```

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
[root@volgalnx010 pigdemos]#
```

10.11.6 ORDER BY

ORDER BY is used to sort a relation based on specific value.

Objective: Display the names of the students in Ascending Order.
Input:
Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = ORDER A BY name;
DUMP B;
```

Output:

```
(1006,Alex,4.5)
(1007,David,4.2)
(1002,Jack,4.0)
(1008,James,4.0)
(1001,John,3.0)
(1001,John,3.0)
(1005,Joshi,3.5)
(1005,Joshi,3.5)
(1004,Scott,4.2)
(1003,Smith,4.5)
[root@volgalnx010 pigdemos]#
```

10.11.7 JOIN

It is used to join two or more relations based on values in the common field. It always performs inner Join.

Objective: To join two relations namely, "student" and "department" based on the values contained in the "rollno" column.
Input:
Student (rollno:int,name:chararray,gpa:float)
Department(rollno:int,deptno:int,deptname:chararray)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = load '/pigdemo/department.tsv' as (rollno:int, deptno:int,deptname:chararray);
C = JOIN A BY rollno, B BY rollno;
DUMP C;
DUMP B;
```

Output:

```
(1001,John,3.0,1001,101,B.E.)
(1001,John,3.0,1001,101,B.E.)
(1002,Jack,4.0,1002,102,B.Tech)
(1003,Smith,4.5,1003,103,M.Tech)
(1004,Scott,4.2,1004,104,MCA)
(1005,Joshi,3.5,1005,105,MBA)
(1005,Joshi,3.5,1005,105,MBA)
(1006,Alex,4.5,1006,101,B.E)
(1007,David,4.2,1007,104,MCA)
(1008,James,4.0,1008,102,B.Tech)
[root@volgalnx010 pigdemos]#
```

10.11.8 UNION

It is used to merge the contents of two relations.

Objective: To merge the contents of two relations "student" and "department".
Input:
Student (rollno:int,name:chararray,gpa:float)
Department(rollno:int,deptno:int,deptname:chararray)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno, name, gp);
B = load '/pigdemo/department.tsv' as (rollno, deptno,deptname);
C = UNION A,B;
STORE C INTO '/pigdemo/uniondemo';
DUMP B;
```

Output:
"Store" is used to save the output to a specified path. The output is stored in two files: part-m-00000 contains "student" content and part-m-00001 contains "department" content.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup
part-m-00000	file	146 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup
part-m-00001	file	114 B	3	128 MB	2015-02-24 17:23	rw-r--r--	root	supergroup

File: /pigdemo/uniondemo/part-m-00000

Goto : /pigdemo/uniondemo go

Go back to dir listing
Advanced view/download options

```
1001    John    3.0
1002    Jack    4.0
1003    Smith   4.5
1004    Scott   4.2
1005    Joshi   3.5
1006    Alex    4.5
1007    David   4.2
1008    James   4.0
1001    John    3.0
1005    Joshi   3.5
```

File: /pigdemo/uniondemo/part-m-00001

Goto : /pigdemo/uniondemo go

Go back to dir listing
Advanced view/download options

```
1001    101     B.E.
1002    102     B.Tech
1003    103     M.Tech
1004    104     MCA
1005    105     MBA
1006    101     B.E
1007    104     MCA
1008    102     B.Tech
```

10.11.9 SPLIT

It is used to partition a relation into two or more relations.

Objective: To partition a relation based on the GPAs acquired by the students.
- GPA = 4.0, place it into relation X.
- GPA is < 4.0, place it into relation Y.

Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
SPLIT A INTO X IF gpa==4.0, Y IF gpa<=4.0;
DUMP X;
```

Output: Relation X

```
(1002,Jack,4.0)
(1008,James,4.0)
[root@volgalnx010 pigdemos]#
```

Output: Relation Y

```
(1001,John,3.0)
(1002,Jack,4.0)
(1005,Joshi,3.5)
(1008,James,4.0)
(1001,John,3.0)
(1005,Joshi,3.5)
[root@volgalnx010 pigdemos]#
```

10.11.10 SAMPLE

It is used to select random sample of data based on the specified sample size.

Objective: To depict the use of ***SAMPLE***.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = SAMPLE A 0.01;
DUMP B;
```

10.12 EVAL FUNCTION

10.12.1 AVG

AVG is used to compute the average of numeric values in a single column bag.

Objective: To calculate the average marks for each student.
Input:
Student (studname:chararray,marks:int)
Act:

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as (studname:chararray,marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname, AVG(A.marks);
DUMP C;
```

Output:

```
({(Jack),(Jack),(Jack),(Jack)},39.75)
({(John),(John),(John),(John)},39.0)
[root@volgalnx010 pigdemos]#
```

Note: You need to use PigStorage function if you wish to manipulate files other than .tsv.

10.12.2 MAX

MAX is used to compute the maximum of numeric values in a single column bag.

Objective: To calculate the maximum marks for each student.
Input:
Student (studname:chararray,marks:int)
Act:

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE A.studname, MAX(A.marks);
DUMP C;
```

Output:

```
({(Jack),(Jack),(Jack),(Jack)},46)
({(John),(John),(John),(John)},45)
[root@volgalnx010 pigdemos]#
```

Note: Similarly, you can try the MIN and the SUM functions as well.

10.12.3 COUNT

COUNT is used to count the number of elements in a bag.

Objective: To count the number of tuples in a bag.
Input:
Student (studname:chararray,marks:int)
Act:

```
A = load '/pigdemo/student.csv' USING PigStorage (',') as (studname:chararray, marks:int);
B = GROUP A BY studname;
C = FOREACH B GENERATE  A. studname,COUNT(A);
DUMP C;
```

Output:

```
({(Jack),(Jack),(Jack),(Jack)},4)
({(John),(John),(John),(John)},4)
[root@volgalnx010 pigdemos]#
```

Note: The default file format of Pig is .tsv file. Use PigStorage() to manipulate files other than .tsv file.

10.13 COMPLEX DATA TYPES

10.13.1 TUPLE

A *TUPLE* is an ordered collection of fields.

Objective: To use the complex data type "Tuple" to load data.
Input:

```
(John,12)      (Jack,13)
(James,7)      (Joseph,5)
(Smith,8)      (Scott,12)
```

Act:

```
A = LOAD '/root/pigdemos/studentdata.tsv' AS (t1:tuple(t1a:chararray,
t1b:int),t2:tuple(t2a:chararray,t2b:int));
B = FOREACH A GENERATE t1.t1a, t1.t1b,t2.$0,t2.$1;
DUMP B;
```

Output:

```
(John,12,Jack,13)
(James,7,Joseph,5)
(Smith,8,Scott,12)
[root@volgalnx010 pigdemos]#
```


Note: You can refer to the field using Positional Notation as shown above. The Positional Notation is denoted by $ sign and the position starts with 0 (e.g., $0).

10.13.2 MAP

MAP represents a key/value pair.

Objective: To depict the complex data type "map".

Input:

John	**[city#Bangalore]**
Jack	**[city#Pune]**
James	**[city#Chennai]**

Act:

```
A = load '/root/pigdemos/studentcity.tsv' Using PigStorage as
(studname:chararray,m:map[chararray]);

B = foreach A generate m#'city' as CityName:chararray;

DUMP B
```

Output:

```
(Bangalore)
(Pune)
(Chennai)
[root@volgalnx010 pigdemos]#
```

10.14 PIGGY BANK

Pig user can use Piggy Bank functions in Pig Latin script and they can also share their functions in Piggy Bank.

Objective: To use Piggy Bank string UPPER function.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
register '/root/pigdemos/piggybank-0.12.0.jar';

A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);

upper = foreach A generate

org.apache.pig.piggybank.evaluation.string.UPPER(name);

DUMP upper;
```


Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(JOSHI)
(ALEX)
(DAVID)
(JAMES)
(JOHN)
(JOSHI)
[root@volgalnx010 pigdemos]#
```

Note: You need to use the "**register**" keyword to use Piggy Bank jar function in your pig script.

10.15 USER-DEFINED FUNCTIONS (UDF)

Pig allows you to create your own function for complex analysis.

Objective: To depict user-defined function.

Java Code to convert name into uppercase:

```
package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;
public class UPPER extends EvalFunc<String>
  {
    public String exec(Tuple input) throws IOException {
      if (input == null || input.size() == 0)
        return null;
        try{
          String str = (String)input.get(0);
          return str.toUpperCase();
        }catch(Exception e){
          throw WrappedIOException.wrap ("Caught exception processing input row ", e);
      }
    }
  }
```

Note: Convert above java class into jar to include this function into your code.

Input:

Student (rollno:int,name:chararray,gpa:float)

Act:

```
register /root/pigdemos/myudfs.jar;
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
B = FOREACH A GENERATE myudfs.UPPER(name);
DUMP B;
```

Output:

```
(JOHN)
(JACK)
(SMITH)
(SCOTT)
(JOSHI)
(ALEX)
(DAVID)
(JAMES)
(JOHN)
(JOSHI)
[root@volgalnx010 pigdemos]#
```

10.16 PARAMETER SUBSTITUTION

Pig allows you to pass parameters at runtime.

Objective: To depict parameter substitution.
Input:
Student (rollno:int,name:chararray,gpa:float)
Act:

```
A = load '$student' as (rollno:int, name:chararray, gpa:float);
DUMP A;
```

Execute:

```
pig –param student=/pigdemo/student.tsv parameterdemo.pig
```

Output:

```
(1001,John,3.0)
(1002,Jack,4.0)
(1003,Smith,4.5)
(1004,Scott,4.2)
(1005,Joshi,3.5)
(1006,Alex,4.5)
(1007,David,4.2)
(1008,James,4.0)
(1001,John,3.0)
(1005,Joshi,3.5)
[root@volgalnx010 pigdemos]#
```

10.17 DIAGNOSTIC OPERATOR

It returns the schema of a relation.

Objective: To depict the use of ***DESCRIBE.***
Input:
Student (rollno:int,name:chararray,gpa:float)

Act:

```
A = load '/pigdemo/student.tsv' as (rollno:int, name:chararray, gpa:float);
DESCRIBE A;
```

Output:

```
A: {rollno: int,name: chararray,gpa: float}
[root@volgalnx010 pigdemos]#
```

10.18 WORD COUNT EXAMPLE USING PIG

Objective: To count the occurrence of similar words in a file.
Input:

```
Welcome to Hadoop Session
Introduction to Hadoop
Introducing Hive
Hive Session
Pig Session
```

Act:

```
lines = LOAD '/root/pigdemos/lines.txt' AS (line:chararray);
words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
grouped = GROUP words BY word;
wordcount = FOREACH grouped GENERATE group, COUNT(words);
DUMP wordcount;
```

Output:

```
(to,2)
(Pig,1)
(Hive,2)
(Hadoop,2)
(Session,3)
(Welcome,1)
(Introducing,1)
(Introduction,1)
[root@volgalnx010 pigdemos]#
```

Note:
TOKENIZE splits the line into a field for each word.
FLATTEN will take the collection of records returned by TOKENIZE and produce a separate record for each one, calling the single field in the record word.

10.19 WHEN TO USE PIG?

Pig can be used in the following situations:

1. When your data loads are time sensitive.
2. When you want to process various data sources.
3. When you want to get analytical insights through sampling.

10.20 WHEN NOT TO USE PIG?

Pig should not be used in the following situations:

1. When your data is completely in the unstructured form such as video, text, and audio.
2. When there is a time constraint because Pig is slower than MapReduce jobs.

10.21 PIG AT YAHOO!

Yahoo uses Pig for two things:

1. **In Pipelines**, to fetch log data from its web servers and to perform cleansing to remove companies interval views and clicks.
2. **In Research**, script is used to test a theory. Pig provides facility to integrate Perl or Python script which can be executed on a huge dataset.

10.22 PIG versus HIVE

Features	Pig	Hive
Used By	Programmers and Researchers	Analyst
Used For	Programming	Reporting
Language	Procedural data flow language	SQL Like
Suitable For	Semi - Structured	Structured
Schema/Types	Explicit	Implicit
UDF Support	YES	YES
Join/Order/Sort	YES	YES
DFS Direct Access	YES (Implicit)	YES (Explicit)
Web Interface	YES	NO
Partitions	YES	NO
Shell	YES	YES

REMIND ME

- Apache Pig is a platform for data analysis. It is an alternative to MapReduce Programming.
- It provides an **engine** for executing **data flows** (how your data should flow). Pig processes data in parallel on the Hadoop cluster.
- It provides a language called "**Pig Latin**" to express data flows.
- The main components of Pig are as follows:
  - Data flow language (**Pig Latin**).
  - Interactive shell where you can type Pig Latin statements (**Grunt**).
  - Pig interpreter and execution engine.
- You can run Pig in two ways:
  - Interactive Mode.
  - Batch Mode.

POINT ME (BOOK)

- Programming Pig, Alan Gates, O'REILY.

CONNECT ME (INTERNET RESOURCES)

- http://pig.apache.org/docs/r0.12.0/index.html
- http://www.edureka.co/blog/introduction-to-pig/
- http://www.edureka.co/blog/pig-vs-hive/

TEST ME

A. Fill Me

1. Pig is a _____ language.
2. In Pig, _____ is used to specify data flow.
3. Pig provides an _____ to execute data flow.
4. _____, _____ are execution modes of Pig.
5. The interactive mode of Pig is _____.
6. _____ and _____ are case sensitive in Pig.
7. _____, _____, _____ are Complex Data Types of Pig.
8. Pig is used in _____ process.

Answers:

1. Scripting
2. Pig Latin
3. Pig Engine
4. Local Mode, MapReduce Mode
5. Grunt
6. Fields and Aliases
7. Bag, Tuple, Map
8. ETL

B. Match Me

Column A	Column B
Map	Hadoop Cluster
Bag	An Ordered Collection of Fields
Local Mode	Collection of Tuples
Tuple	Key/Value Pair
MapReduce Mode	Local File System

Answers:

Column A	Column B
Map	Key/Value Pair
Bag	Collection of Tuples
Local Mode	Local File System
Tuple	An Ordered Collection of Fields
MapReduce Mode	Hadoop Cluster

C. True or False

1. PigStorage() function is case sensitive.
2. Local Mode is the default mode of Pig.
3. DISTINCT Keyword removes duplicate fields.
4. LIMIT keyword is used to display limited number of tuples in Pig.
5. ORDER BY is used for sorting.

Answers:

1. True
2. False
3. False
4. True
5. True

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1: SPLIT

Objective: To learn about SPLIT relational operator.

Problem Description:
Write a Pig Script to split customers for reward program based on their life time values.

Input:

Customers	Life Time Value
Jack	25000
Smith	8000
David	35000
John	15000
Scott	10000
Joshi	28000
Ajay	12000
Vinay	30000
Joseph	21000

- If Life Time Value is >1000 and <= 2000 → Sliver Program.
- If Life Time Value is >20000 → Gold Program.

ASSIGNMENT 2: GROUP

Objective: To learn about GROUP relational operator.

Problem Description:
Create a data file for below schemas:

- **Order:** CustomerId, ItemId, ItemName, OrderDate, DeliveryDate
- **Customer:** CustomerId, CustomerName, Address, City, State, Country

1. Load Order and Customer Data.
2. Write a Pig Latin Script to determine number of items bought by each customer.

ASSIGNMENT 3: COMPLEX DATA TYPE – BAG

Objective: To learn complex data type – bag in Pig.

Problem Description:

1. Create a file which contains bag dataset as shown below.

User ID	From	To
user1001	user1001@sample.com	{(user003@sample.com),(user004@sample.com), (user006@sample.com)}
user1002	user1002@sample.com	{(user005@sample.com), (user006@sample.com)}
user1003	user1003@sample.com	{(user001@sample.com),(user005@sample.com)}

2. Write a Pig Latin statement to display the names of all users who have sent emails and also a list of all the people that they have sent the email to.
3. Store the result in a file.