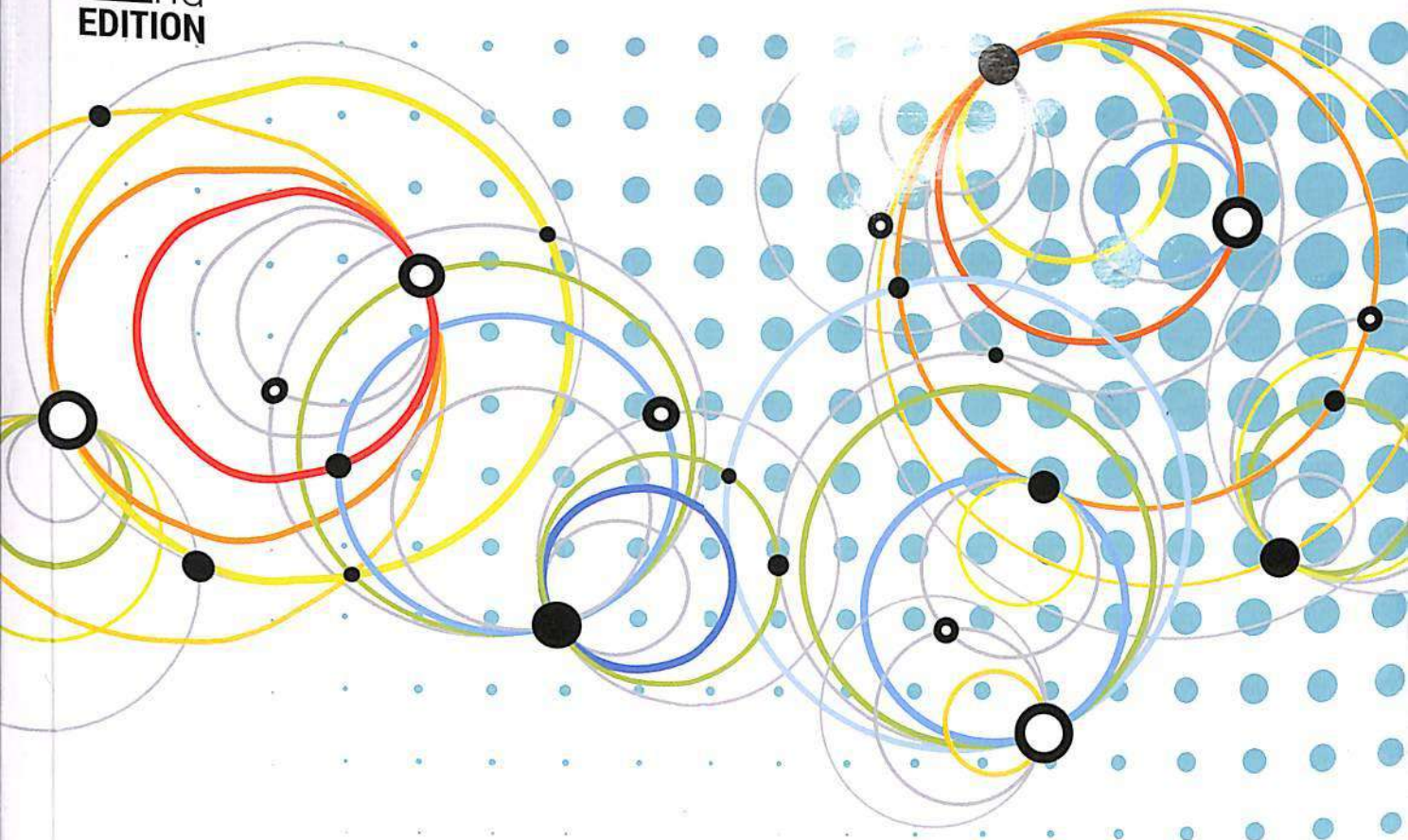
2nd EDITION



BIG DATA AND ANALYTICS

Seema Acharya
Subhashini Chellappan

WILEY

2nd EDITION

BIG DATA AND ANALYTICS

Seema Acharya
Infosys Limited

Subhashini Chellappan

WILEY

Big Data and Analytics

SECOND EDITION

Copyright © 2019 by Wiley India Pvt. Ltd., 4436/7, Ansari Road, Daryaganj, New Delhi-110002.

Cover Image: © aleksandarvelasevic/Getty Images

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or scanning without the written permission of the publisher.

Limits of Liability: While the publisher and the author have used their best efforts in preparing this book, Wiley and the author make no representation or warranties with respect to the accuracy or completeness of the contents of this book, and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose. There are no warranties which extend beyond the descriptions contained in this paragraph. No warranty may be created or extended by sales representatives or written sales materials. The accuracy and completeness of the information provided herein and the opinions stated herein are not guaranteed or warranted to produce any particular results, and the advice and strategies contained herein may not be suitable for every individual. Neither Wiley India nor the author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Disclaimer: The contents of this book have been checked for accuracy. Since deviations cannot be precluded entirely, Wiley or its author cannot guarantee full agreement. As the book is intended for educational purpose, Wiley or its author shall not be responsible for any errors, omissions or damages arising out of the use of the information contained in the book. This publication is designed to provide accurate and authoritative information with regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services.

Trademarks: All brand names and product names used in this book are trademarks, registered trademarks, or trade names of their respective holders. Wiley is not associated with any product or vendor mentioned in this book.

Other Wiley Editorial Offices:

John Wiley & Sons, Inc. 111 River Street, Hoboken, NJ 07030, USA

Wiley-VCH Verlag GmbH, Pappellaee 3, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 1 Fusionopolis Walk #07-01 Solaris, South Tower, Singapore 138628

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada, M9W 1L1

First Edition: 2015

Second Edition : 2019

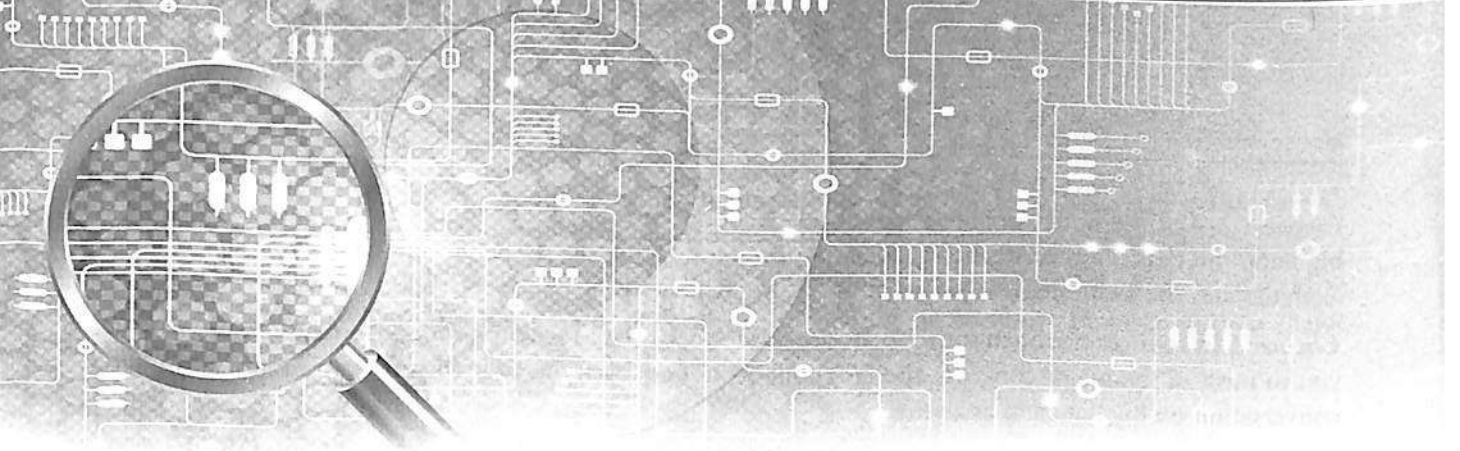
Reprint: 2022

ISBN: 978-81-265-7951-8

ISBN: 978-81-265-8836-7 (ebk)

www.wileyindia.com

Printed at: Printways



Preface

The last few years have been witness to a burgeoning growth of data. We have heard it being called Big Data! So what really is this big data? Big data is an evolving term used to describe any voluminous amount of structured, semi-structured and unstructured data that has the potential to be mined for information. There is data everywhere from the sensors that gather weather information, to the likes, posts and comments on social media sites, to digital pictures, audios and videos that get circulated, to the conversations in a chat room, etc. All this and more is big data.

Need for this Book

We felt the need to compose a book, egged on by the enthusiasm and inquisitiveness of the students and instructors fraternity alike. A book which can take the readers through an easy comprehension of the big data technology landscape. Ours is an attempt to cover a plethora of technologies from NoSQL databases such as MongoDB and Cassandra to components of the Hadoop Ecosystem such as MapReduce, Pig and Hive to delving into analytics with association rule mining on one hand and decision trees on the other hand.

The Audience

This book is for all interested in learning about Big Data, Hadoop and Analytics. The only criteria is the willingness to learn and the ability to stretch yourself in learning to limits that you have not done before. The book is for all those who are new to big data irrespective of the field/background that you come from.

The book will be equally useful to an engineering graduate as it would be to a management graduate. The book has been designed and crafted such that it caters to the knowledge requirements of an IT person as well as a business user with ease.

Organization of the Book

This book has a total of 14 chapters. Here is a sneak peek into the chapters of our book...

Chapters 1–4 of the book provide a basic understanding of the types of digital data, the characteristics of big data, the challenges confronting the enterprises embracing big data, the sudden hype around big data analytics and the technologies that make up the big data landscape.

Chapter 5 introduces the open source software framework called Hadoop. We have attempted to introduce you to most of the major concepts and components to empower you to hold your own in any meaningful conversation on big data and analytics.

Chapters 6 and 7 introduce you to the world of NoSQL databases. We have chosen MongoDB, the document-oriented database, and Cassandra, the wide column store, to get you a feel of NoSQL databases. In explaining the NoSQL databases, we have built on the familiarity that the readers will have with RDBMS (Relational Database Management System).

Chapter 8 introduces you to the nitty-gritties of MapReduce Programming. The merits and challenges have been dealt with for a clearer appreciation.

Chapters 9 and 10 cover two major components of the Hadoop Ecosystem, namely "Pig" and "Hive".

Chapter 11 introduces to you an open source tool to draw out reports by pulling data from NoSQL databases.

Chapter 12 is focused on introducing you to the world of machine learning and analysis with algorithms under both supervised and unsupervised learning categories.

Chapter 13 is focused on bringing out the differences between various Hadoop ecosystem components for an easy lookup and remembrance. It will be good to read this chapter sequentially for better absorption. Starting with data warehouse versus data lakes, HDFS versus the first non-batch component – Hbase, it builds further to explain the differences between HDFS and RDBMS, then goes onto to highlight differences of MapReduce with Pig, Spark and finally delves into the differences between Pig and Hive.

Chapter 14 discusses the big data trends in 2019 and beyond. The years ahead will see an increase in the adoption of open-source technologies. Hadoop is and will remain fundamental, although there will be increased usage of the in-memory Spark. The years ahead will also awake to the container(ed) revolution. The last half a decade has been a witness to the commoditization of visualization. The rising wave of IoT (Internet of Things) will lead to processing being done on the edge of the network before moving it to the central data center in the cloud. The world will witness the power of empowered computing – edge and quantum. It is time to utilize and draw value/insight from the abundant dark data. Also bots will mature and get smarter in the coming years.

Glossary: A glossary of terms frequently used in the big data and analytics parlance is given at the end of the book. Although we strive to define terms as we introduce them in this book, we think you'll find the glossary a useful resource.

To get most out of this Book

We have included sections such as "POINT ME", "CONNECT ME", "TEST ME" to enable you to further your learning and comprehension.

The section "*POINT ME*" provides a list of books that you as a reader should check out to further your learning.

The section "*CONNECT ME*" provides a list of reference links which will feed you with good content on topics covered in the chapter.

The section "*TEST ME*" has a gamut of self-assessments such as "Crossword" puzzles, "Fill in the blanks", "Match the columns", etc. We have provided solved and unsolved exercises to better your learning.

There are *HANDS-ON ASSIGNMENTS* provided with MongoDB, Cassandra, MapReduce, Pig, Hive and JasperReports. We sincerely urge you to attempt these to gain good hands-on practice on these major technologies.

Next Steps...

We have endeavored to create an overview of big data and introduced you to all its significant components. We recommend you to read the book from cover to cover, but if you are not that kind of person, we have made an attempt to keep the chapters self-contained so that you can go straight to the topics that interest you most.

Whichever approach you may choose, we wish you well!

Available with the Book (www.wileyindia.com)

We have put together an installation guide to help our learners with easy steps to install and configure a Hadoop cluster. The steps to setting up the components of the Hadoop ecosystem such as MapReduce, Pig and Hive have also been explained in easy, DIY (Do It Yourself) steps.

We have provided a Microsoft Access Database (.accdb) and a text file on which we have based an assignment that when attempted and solved will surely challenge and satiate you.

A Quick Word for the Instructors' Fraternity

Attention has been paid in arriving at the sequence of chapters and also to the flow of topics within each chapter. This is done particularly with an objective to assist our fellow instructors and academicians in carving out a syllabi from the Table of Contents (TOC) of the book. The complete TOC can qualify as the syllabi for a semester or if the college has an existing syllabus on big data and analytics, a few chapters can be added to the syllabi to make it more robust. We leave it to your discretion on how you wish to use the same for your students.

We have ensured that each tool/component discussed in the book is with adequate hands-on content to enable you to teach better and provide ample hands-on practice to your students.

The easy-to-follow installation guide provided on the website should help you set up the lab environment for practice.

We have also provided Instructor Resources (IR) that can be procured directly from our publisher, Wiley India by visiting their website or writing to acadmktg@wiley.com. These Instructor Resources are presentation decks (one for each chapter) which can be taken to the class directly or can be customized as per your requirements.

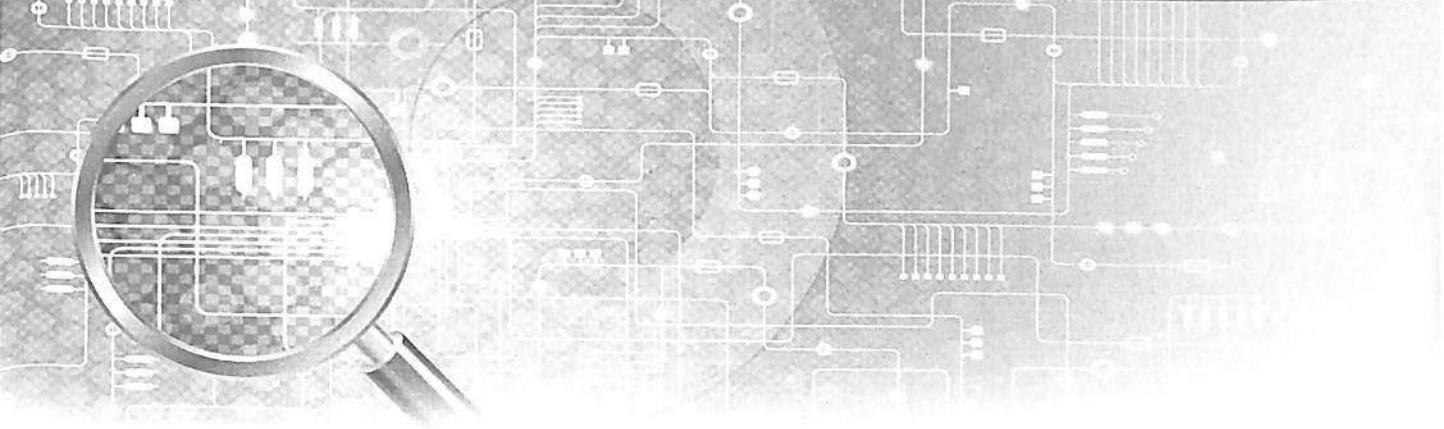
Connect with Authors

To stay connected with the students and instructors fraternity, we run a group on LinkedIn titled, "*Exploring big data and analytics*". Join us to discuss, share and learn!!!

Happy Learning!!!

Seema Acharya

Subhashini Chellappan



Acknowledgements

The making of the book was like a journey that we had undertaken for several months. We had our families, friends, colleagues, and well-wishers onboard this journey and we wish to express our heartfelt gratitude to each one of them. Without their unflinching support and affection, we could not have pulled it off.

We are grateful to the student and teacher community who kept us on our toes with their constant bombardment of queries which prompted us to learn more, simplify our learnings and findings, and place them neatly in the book. This book is for them.

We wish to thank our friends – the practitioners from the field for filling us in on the latest in the big data field and sharing with us valuable insights on the best practices and methodologies followed therein.

A special thanks to R N Prasad for his encouragement and vigilant review.

We have been fortunate to have the support of our teams who sometimes knowingly and at other times unknowingly contributed to the making of the book by lending us their unwavering support.

We consider ourselves very fortunate for the editorial assistance provided by Wiley India. We wish to acknowledge and appreciate Meenakshi Sehrawat, Associate Publisher and her team of associates who adeptly guided us through the entire process of preparation and publication. Appreciation is also due to Rakesh Poddar and his team for working with us through the entire production process.

And finally we can never sufficiently thank our families and friends who have been our pillars of strength, our stimulus, and our soundboards all through the process, and endured patiently our crazy schedules as we assembled the book.

Author Profile

Seema Acharya

Seema Acharya is a Senior Lead Principal with the Education, Training and Assessment department of Infosys Limited. She is a technology evangelist, a learning strategist, and an author with over 15+ years of IT experience in learning/education services. She has designed and delivered several large-scale competency development programs across the globe involving organizational competency need analysis, conceptualization, design, development and deployment of competency development programs. She is an educator by choice and vocation, and has rich experience in both academia and the software industry.

She is also the author of the following books:

1. "Fundamentals of Business Analytics", ISBN: 978-81-265-3203-2, publisher – Wiley India.
2. "Pro Tableau – A Step by Step Guide", ISBN: 978-1484223512, publisher – Apress.
3. "Data Analytics using R", ISBN: 9789352605248, publisher – McGraw Hill Higher Education Society (2018).

She has co-authored a paper on "Collaborative Engineering Competency Development" for ASEE (American Society for Engineering Education). She holds the patent on "Method and System for Automatically Generating Questions for a Programming Language".

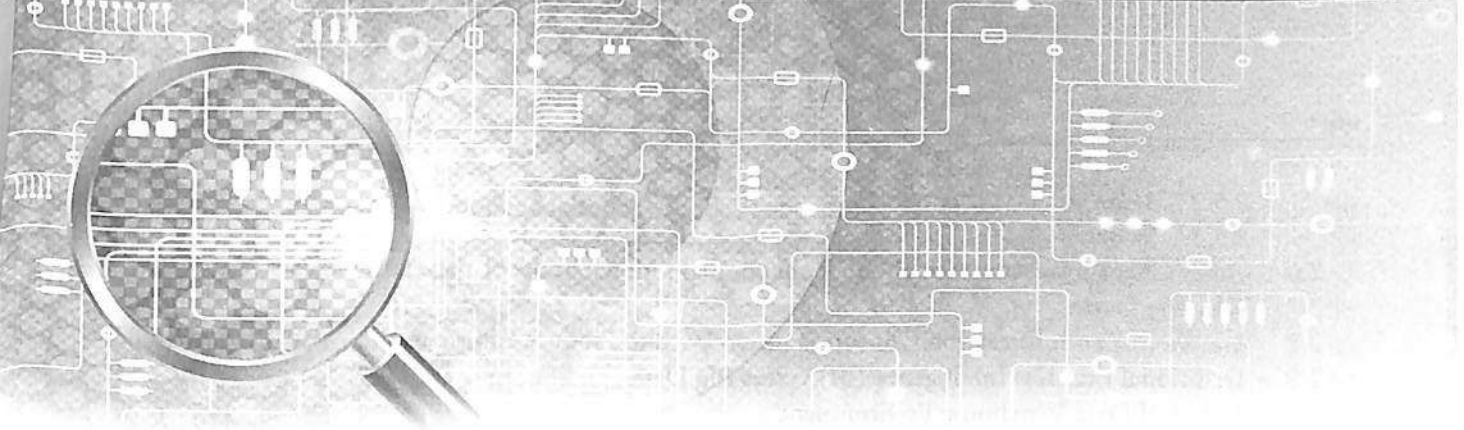
Her areas of interest and expertise are centered on Business Intelligence, Big Data and Analytics, technologies such as Data Warehousing, Data Mining, Data Analytics, Text Mining and Data Visualization.

She is passionate about exploring new paradigms of learning and also dabbles into creating e-learning content to facilitate learning anytime and anywhere.

Subhashini Chellappan

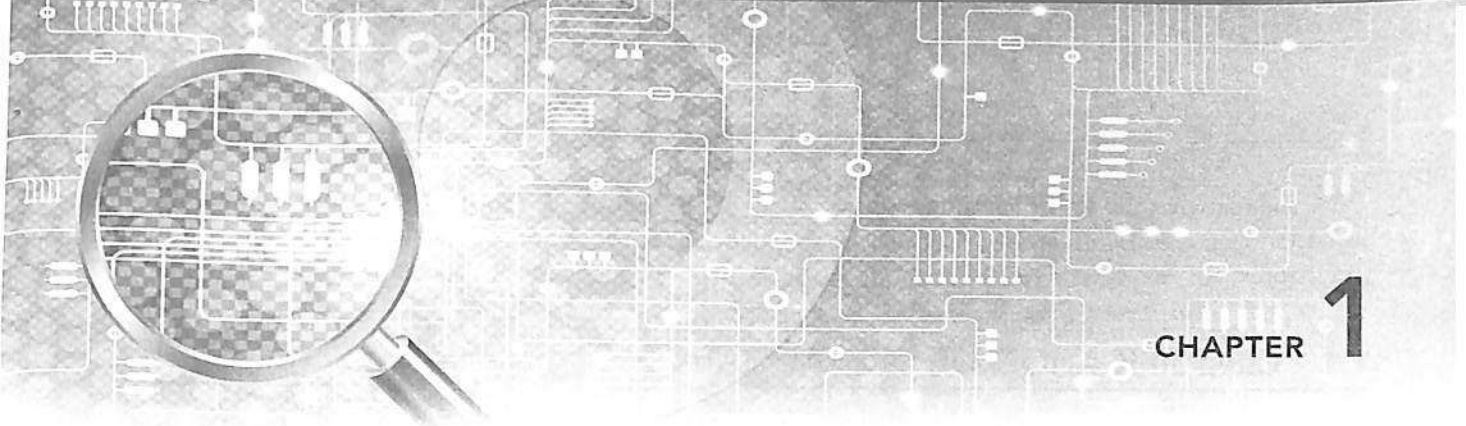
Subhashini Chellappan has rich experience in both academia and the software industry. She has published couple of papers in various Journals and Conferences.

Her areas of interest and expertise are centered on Business Intelligence, Big Data and Analytics technologies such as Hadoop, NoSQL Databases, Spark and Machine Learning.



Contents

Chapter 10 Introduction to Pig 269



CHAPTER 1

Types of Digital Data

BRIEF CONTENTS

"In God we trust, all others must bring data."

– W. Edwards Deming

WHAT'S IN STORE?

Irrespective of the size of the enterprise (big or small), data continues to be a precious and irreplaceable asset. Data is present internal to the enterprise and also exists outside the four walls and firewalls of the enterprise. Data is present in homogeneous sources as well as in heterogeneous sources. The need of the hour is to understand, manage, process, and take the data for analysis to draw valuable insights.

Data → Information

Information → Insights

This chapter is a "must read" for first-time learners interested in understanding the role of data in business intelligence and business analysis and businesses at large. This chapter will introduce you to the various formats of digital data (structured, semi-structured, and unstructured data), the sources of each format of data, the issues with the terminology of unstructured data, etc.

We suggest you refer to the learning resources suggested at the end of this chapter and also attempt all the exercises to get a grip on this topic. We suggest you make your own notes/bookmarks while reading through the chapter.

1.1 CLASSIFICATION OF DIGITAL DATA

As depicted in Figure 1.1, digital data can be broadly classified into structured, semi-structured, and unstructured data.

1. **Unstructured data:** This is the data which does not conform to a data model or is not in a form which can be used easily by a computer program. About 80–90% data of an organization is in this format; for example, memos, chat rooms, PowerPoint presentations, images, videos, letters, researches, white papers, body of an email, etc.
2. **Semi-structured data:** This is the data which does not conform to a data model but has some structure. However, it is not in a form which can be used easily by a computer program; for example, emails, XML, markup languages like HTML, etc. Metadata for this data is available but is not sufficient.
3. **Structured data:** This is the data which is in an organized form (e.g., in rows and columns) and can be easily used by a computer program. Relationships exist between entities of data, such as classes and their objects. Data stored in databases is an example of structured data.

Ever since the 1980s most of the enterprise data has been stored in relational databases complete with rows/records/tuples, columns/attributes/fields, primary keys, foreign keys, etc. Over a period of time Relational Database Management System (RDBMS) matured and the RDBMS, as they are available today, have become more robust, cost-effective, and efficient. We have grown comfortable working with RDBMS – the storage, retrieval, and management of data has been immensely simplified. The data held in RDBMS is typically structured data. However, with the Internet connecting the world, data that existed beyond one's enterprise started to become an integral part of daily transactions. This data grew by leaps and bounds so much so that it became difficult for the enterprises to ignore it. All of this data was not structured. A lot of it was unstructured. In fact, Gartner estimates that almost 80% of data generated in any enterprise today is unstructured data. Roughly around 10% of data is in the structured and semi-structured category. Refer Figure 1.2.

1.1.1 Structured Data

Let us begin with a very basic question – When do we say that the data is structured? The simple answer is when data conforms to a pre-defined schema/structure we say it is structured data.

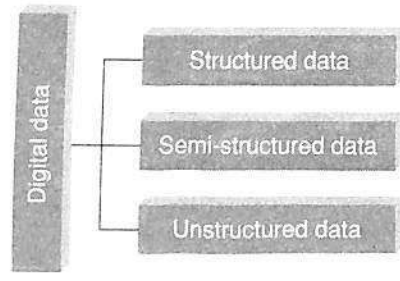


Figure 1.1 Classification of digital data.

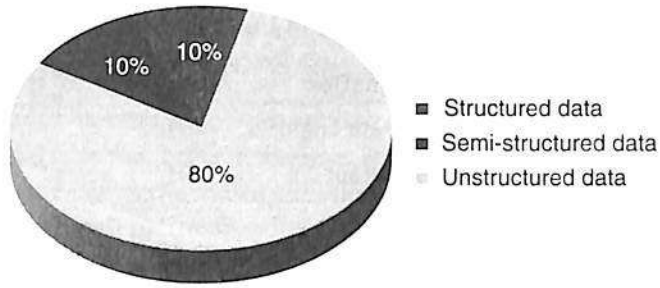


Figure 1.2 Approximate percentage distribution of digital data.

Think structured data, and think data model – a model of the types of business data that we intend to store, process, and access. Let us discuss this in the context of an RDBMS. Most of the structured data is held in RDBMS. An RDBMS conforms to the relational data model wherein the data is stored in rows/columns. Refer Table 1.1.

The number of rows/records/tuples in a relation is called the *cardinality of a relation* and the number of columns is referred to as the *degree of a relation.*

The first step is the design of a relation/table, the fields/columns to store the data, the type of data that will be stored [number (integer or real), alphabets, date, Boolean, etc.]. Next we think of the constraints that we would like our data to conform to (constraints such as UNIQUE values in the column, NOT NULL values in the column, a business constraint such as the value held in the column should not drop below 50, the set of permissible values in the column such as the column should accept only "CS", "IS", "MS", etc., as input).

To explain further, let us design a table/relation structure to store the details of the employees of an enterprise. Table 1.2 shows the structure/schema of an "Employee" table in a RDBMS such as Oracle.

Table 1.2 is an example of a good structured table (complete with table name, meaningful column names with data types, data length, and the relevant constraints) with absolute adherence to relational data model.

Table 1.1 A relation/table with rows and columns

	Column 1	Column 2	Column 3	Column 4
Row 1				

Table 1.2 Schema of an "Employee" table in a RDBMS such as Oracle

Column Name	Data Type	Constraints
EmpNo	Varchar(10)	PRIMARY KEY
EmpName	Varchar(50)	
Designation	Varchar(25)	NOT NULL
DeptNo	Varchar(5)	
ContactNo	Varchar(10)	NOT NULL

Table 1.3 Sample records in the "Employee" table

EmpNo	EmpName	Designation	DeptNo	ContactNo
E101	Allen	Software Engineer	D1	0999999999
E102	Simon	Consultant	D1	0777777777

It goes without saying that each record in the table will have exactly the same structure. Let us take a look at a few records in Table 1.3.

The tables in an RDBMS can also be related. For example, the above "Employee" table is related to the "Department" table on the basis of the common column, "DeptNo". It is not mandatory for the two tables that are related to have exactly the same name for the common column. On the contrary, the two tables are related on the basis of values held within the column, "DeptNo". Given in Figure 1.3 is a depiction of referential integrity constraint (primary – foreign key) with the "Department" table being the referenced table and "Employee" table being the referencing table.

1.1.1.1 Sources of Structured Data

If your data is highly structured, one can look at leveraging any of the available RDBMS [Oracle Corp. – Oracle, IBM – DB2, Microsoft – Microsoft SQL Server, EMC – Greenplum, Teradata – Teradata, MySQL (open source), PostgreSQL (advanced open source), etc.] to house it. Refer Figure 1.4. These databases are typically used to hold transaction/operational data generated and collected by day-to-day business activities. In other words, the data of the On-Line Transaction Processing (OLTP) systems are generally quite structured.

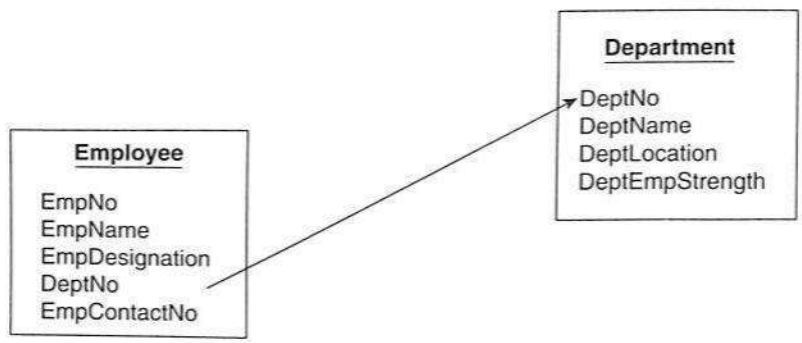


Figure 1.3 Relationship between "Employee" and "Department" tables.

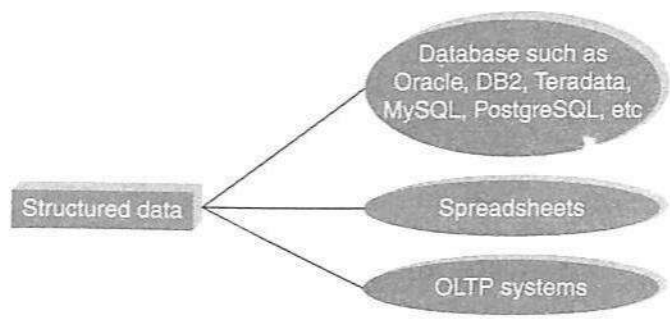


Figure 1.4 Sources of structured data.

1.1.1.2 Ease of Working with Structured Data

Structured data provides the ease of working with it. Refer Figure 1.5. The ease is with respect to the following:

1. **Insert/update/delete:** The Data Manipulation Language (DML) operations provide the required ease with data input, storage, access, process, analysis, etc.
2. **Security:** How does one ensure the security of information? There are available staunch encryption and tokenization solutions to warrant the security of information throughout its lifecycle. Organizations are able to retain control and maintain compliance adherence by ensuring that only authorized individuals are able to decrypt and view sensitive information.
3. **Indexing:** An index is a data structure that speeds up the data retrieval operations (primarily the SELECT DML statement) at the cost of additional writes and storage space, but the benefits that ensue in search operation are worth the additional writes and storage space.
4. **Scalability:** The storage and processing capabilities of the traditional RDBMS can be easily scaled up by increasing the horsepower of the database server (increasing the primary and secondary or peripheral storage capacity, processing capacity of the processor, etc.).
5. **Transaction processing:** RDBMS has support for Atomicity, Consistency, Isolation, and Durability (ACID) properties of transaction. Given next is a quick explanation of the ACID properties:
   - ***Atomicity:*** A transaction is atomic, means that either it happens in its entirety or none of it at all.
   - ***Consistency:*** The database moves from one consistent state to another consistent state. In other words, if the same piece of information is stored at two or more places, they are in complete agreement.
   - ***Isolation:*** The resource allocation to the transaction happens such that the transaction gets the impression that it is the only transaction happening in isolation.
   - ***Durability:*** All changes made to the database during a transaction are permanent and that accounts for the durability of the transaction.

1.1.2 Semi-Structured Data

Semi-structured data is also referred to as self-describing structure. Refer Figure 1.6. It has the following features:

1. It does not conform to the data models that one typically associates with relational databases or any other form of data tables.
2. It uses tags to segregate semantic elements.

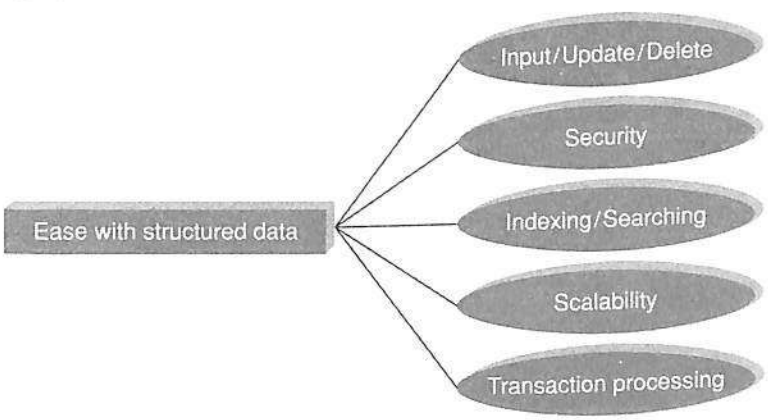


Figure 1.5 Ease of working with structured data.

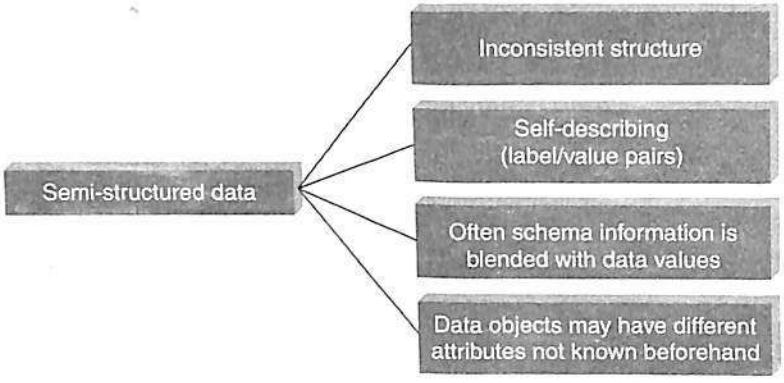


Figure 1.6 Characteristics of semi-structured data.

3. Tags are also used to enforce hierarchies of records and fields within data.
4. There is no separation between the data and the schema. The amount of structure used is dictated by the purpose at hand.
5. In semi-structured data, entities belonging to the same class and also grouped together need not necessarily have the same set of attributes. And if at all, they have the same set of attributes, the order of attributes may not be similar and for all practical purposes it is not important as well.

1.1.2.1 Sources of Semi-Structured Data

Amongst the sources for semi-structured data, the front runners are "XML" and "JSON" as depicted in Figure 1.7.

1. **XML:** eXtensible Markup Language (XML) is hugely popularized by web services developed utilizing the Simple Object Access Protocol (SOAP) principles.
2. **JSON:** Java Script Object Notation (JSON) is used to transmit data between a server and a web application. JSON is popularized by web services developed utilizing the Representational State Transfer (REST) – an architecture style for creating scalable web services. MongoDB (open-source, distributed, NoSQL, documented-oriented database) and Couchbase (originally known as Membase, open-source, distributed, NoSQL, document-oriented database) store data natively in JSON format.

An example of HTML is as follows:

```
<HTML>
<HEAD>
<TITLE>Place your title here</TITLE>
</HEAD>
<BODY BGCOLOR="FFFFFF">
```

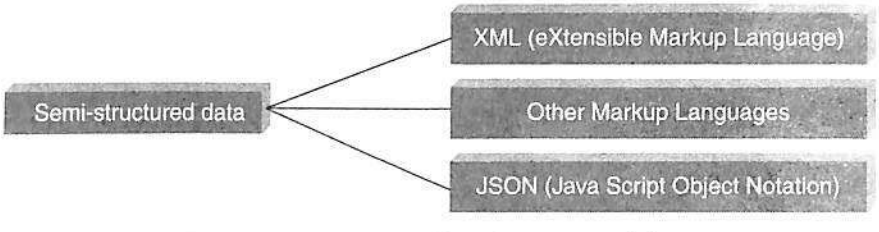


Figure 1.7 Sources of semi-structured data.

```
<CENTER><IMG SRC="clouds.jpg" ALIGN="BOTTOM"></CENTER>
<HR>
<a href="http://bigdatauniversity.com">Link Name</a>
<H1>this is a Header</H1>
<H2>this is a sub Header</H2>
Send me mail at <a href="mailto:support@yourcompany.com">
support@yourcompany.com</a>.
<P>a new paragraph!
<P><B>a new paragraph!</B>
<BR><B><I>this is a new sentence without a paragraph break, in bold italics.</I></B>
<HR>
</BODY>
</HTML>
```

Sample JSON document

```
{
_id:9,
BookTitle: "Fundamentals of Business Analytics",
AuthorName: "Seema Acharya",
Publisher: "Wiley India",
YearofPublication: "2011"
}
```

1.1.3 Unstructured Data

Unstructured data does not conform to any pre-defined data model. In fact, to explain things a little more, let us take a closer look at the various kinds of text available and the possible structure associated with it. As can be seen from the examples quoted in Table 1.4, the structure is quite unpredictable. In Figure 1.8 we look at the other sources of unstructured data.

1.1.3.1 Issues with "Unstructured" Data

Although unstructured data is known NOT to conform to a pre-defined data model or be organized in a pre-defined manner, there are incidents wherein the structure of the data (placed in the unstructured category) can still be implied. As mentioned in Figure 1.9, there could be few other reasons behind placing data in the unstructured category despite it having some structure or being highly structured.

There are situations where people argue that a text file should be in the category of semi-structured data and not unstructured data. Let us look at where they are coming from. Well, the text file does have a name,

Table 1.4 Few examples of disparate unstructured data

Twitter message	Feeling miffed ☹. Victim of twishing.
Facebook post	LOL. C ya. BFN
Log files	127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I; Nav)"
Email	Hey Joan, possible to send across the first cut on the Hadoop chapter by Friday EOD or maybe we can meet up over a cup of coffee. Best regards, Tom

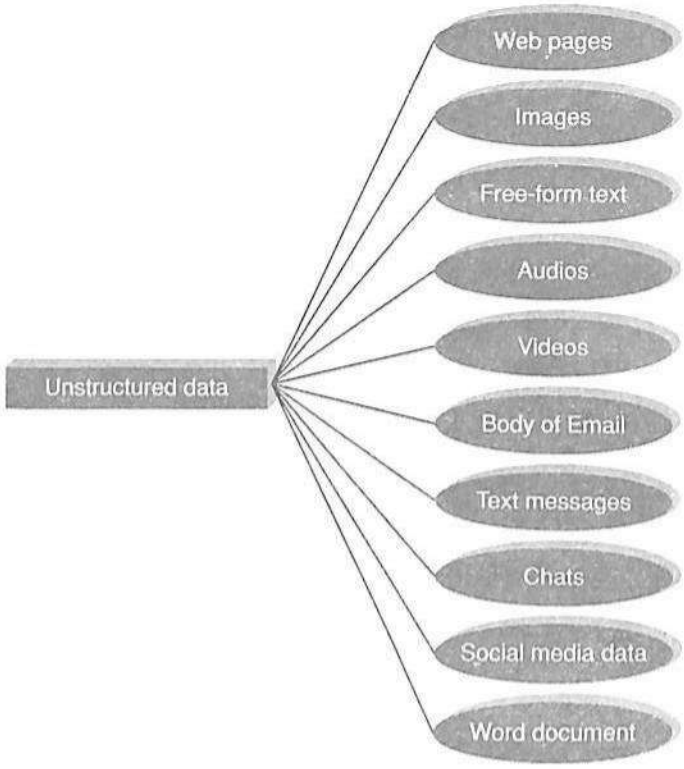


Figure 1.8 Sources of unstructured data.

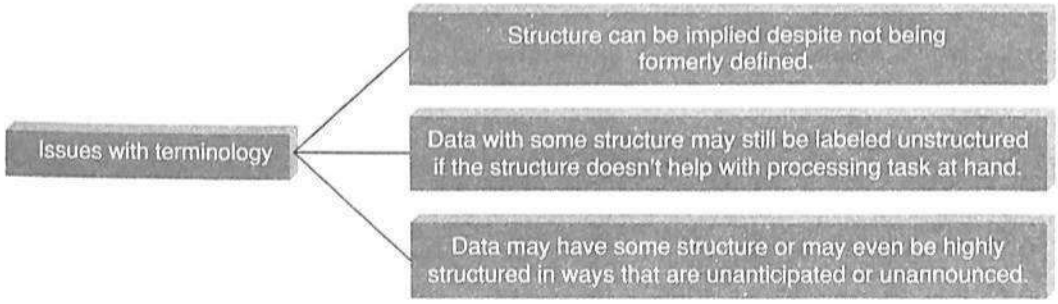


Figure 1.9 Issues with terminology of unstructured data.

one can easily look at the properties to get information such as the owner of the file, the date on which the file was created, the size of the file, etc. Okay, we do have little metadata. But when it comes to analysis, we are more concerned with the content of the text file rather than the name or any of the other properties. In fact, the other properties may not in any way contribute to the processing/analysis task at hand. Therefore, it is fair to place it in the unstructured data category.

1.1.3.2 How to Deal with Unstructured Data?

Today, unstructured data constitutes approximately 80% of the data that is being generated in any enterprise. The balance is clearly shifting in favor of unstructured data as shown in Figure 1.10. It is such a big percentage that it cannot be ignored. Figure 1.11 states a few ways of dealing with unstructured data.

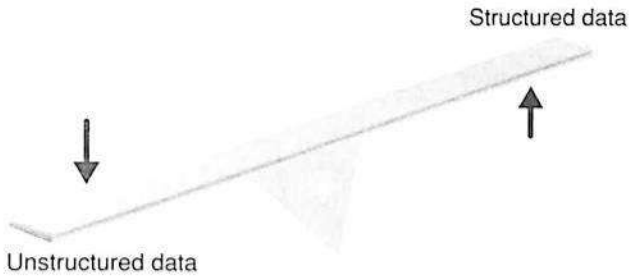


Figure 1.10 Unstructured data clearly constitutes a major percentage of enterprise data.

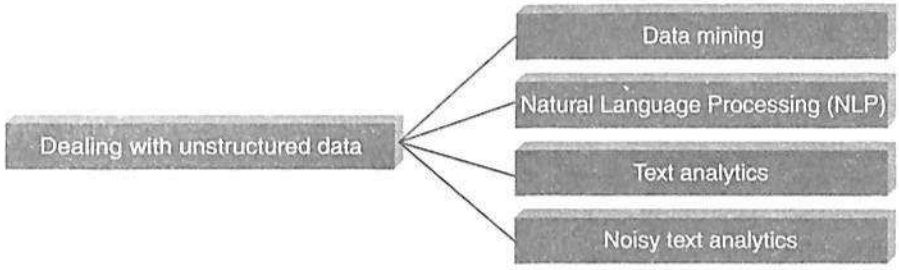


Figure 1.11 Dealing with unstructured data.

The following techniques are used to find patterns in or interpret unstructured data:

1. **Data mining:** First, we deal with large data sets. Second, we use methods at the intersection of artificial intelligence, machine learning, statistics, and database systems to unearth consistent patterns in large data sets and/or systematic relationships between variables. It is the analysis step of the "knowledge discovery in databases" process.

   Few popular data mining algorithms are as follows:
   - ***Association rule mining:*** It is also called "market basket analysis" or "affinity analysis". It is used to determine "What goes with what?" It is about when you buy a product, what is the other product that you are likely to purchase with it. For example, if you pick up bread from the grocery, are you likely to pick eggs or cheese to go with it.
   - ***Regression analysis:*** It helps to predict the relationship between two variables. The variable whose value needs to be predicted is called the dependent variable and the variables which are used to predict the value are referred to as the independent variables.

PICTURE THIS...

You are interested in purchasing real estate. You have been looking at a few good sites. You have come to the conclusion that cost of the real estate depends on the location (outskirts or prime locale), the amenities provided by the builder (joggers track, senior citizen zone, gymnasium, swimming pools, etc.), the built up area, etc. The cost of the real estate is the dependent variable and the location, amenities, built-up area are called the independent variables.

Table 1.5 Sample records depicting learners' preferences for modes of learning

	Learning using Audios	Learning using Videos	Textual Learners
User 1	Yes	Yes	No
User 2	Yes	Yes	Yes
User 3	Yes	Yes	No
User 4	Yes	?	?

- ***Collaborative filtering:*** It is about predicting a user's preference or preferences based on the preferences of a group of users. For example, take a look at Table 1.5.
  We are looking at predicting whether User 4 will prefer to learn using videos or is a textual learner depending on one or a couple of his or her known preferences. We analyze the preferences of similar user profiles and on the basis of it, predict that User 4 will also like to learn using videos and is not a textual learner.

2. **Text analytics or text mining:** Compared to the structured data stored in relational databases, text is largely unstructured, amorphous, and difficult to deal with algorithmically. Text mining is the process of gleaning high quality and meaningful information (through devising of patterns and trends by means of statistical pattern learning) from text. It includes tasks such as text categorization, text clustering, sentiment analysis, concept/entity extraction, etc.
3. **Natural language processing (NLP):** It is related to the area of human computer interaction. It is about enabling computers to understand human or natural language input.
4. **Noisy text analytics:** It is the process of extracting structured or semi-structured information from noisy unstructured data such as chats, blogs, wikis, emails, message-boards, text messages, etc. The noisy unstructured data usually comprises one or more of the following: Spelling mistakes, abbreviations, acronyms, non-standard words, missing punctuation, missing letter case, filler words such as "uh", "um", etc.
5. **Manual tagging with metadata:** This is about tagging manually with adequate metadata to provide the requisite semantics to understand unstructured data.
6. **Part-of-speech tagging:** It is also called POS or POST or grammatical tagging. It is the process of reading text and tagging each word in the sentence as belonging to a particular part of speech such as "noun", "verb", "adjective", etc.
7. **Unstructured Information Management Architecture (UIMA):** It is an open source platform from IBM. It is used for real-time content analytics. It is about processing text and other unstructured data to find latent meaning and relevant relationship buried therein. Read up more on UIMA at the link: http://www.ibm.com/developerworks/data/downloads/uima/

REMIND ME

- ***Structured data:*** It conforms to a data model. For example, RDBMS conforms to relational data model. It has a pre-defined schema.
- ***Semi-structured data:*** For this format of data, little metadata is available, but is insufficient. Semi-structured data have a self-describing structure. There is little or no separation between data and schema.

- ***Unstructured data:*** This data is growing by the day and growing by leaps and bounds. It has innumerable sources such as human generated (social media data, emails, word documents, presentations, audio and video files that we create and share every day, etc.) and machine generated data (sensors, web server logs, call data records, etc.).

POINT ME (BOOK)

- Chapter 2: Types of Digital Data, "Fundamentals of Business Analytics", Wiley India; Authors – RN Prasad and Seema Acharya, 2011.

CONNECT ME (INTERNET RESOURCES)

- http://data-magnum.com/the-big-deal-about-big-data-whats-inside-structured-unstructured-and-semi-structured-data/
- http://www.webopedia.com/TERM/S/structured_data.html
- http://en.wikipedia.org/wiki/UIMA
- Matching unstructured data and structured data by Bill Inmon: http://www.tdan.com/view-articles/5009
- Semi-structured data analytics: Relational or Hadoop platform? – IBM: http://www.ibmbigdatahub.com/blog/semi-structured-data-analytics-relational-or-hadoop-platform-part-1

TEST ME

A. Place Me in the Basket

Structured	Unstructured	Semi-Structured

Following words are to be placed in the relevant basket:

Email
MS Access
Images
Database
Chat conversations
Relations/Tables
Facebook
Videos
MS Excel
XML

Answer:

Structured	Unstructured	Semi-Structured
MS Access	Email	XML
Database	Images	
Relations/Tables	Chat conversations	
MS Excel	Facebook	
	Videos	

B. Match the Following

1.

Column A	Column B
NLP	Content analytics
Text analytics	Text messages
UIMA	Chats
Noisy unstructured data	Text mining
Data mining	Comprehend human or natural language input
Noisy unstructured data	Uses methods at the intersection of statistics, AI, machine learning & DBs
IBM	UIMA

Answer:

Column A	Column B
NLP	Comprehend human or natural language input
Text analytics	Text mining
UIMA	Content analytics
Noisy unstructured data	Text messages
Data mining	Uses methods at the intersection of statistics, AI, machine learning & DBs
Noisy unstructured data	Chats
IBM	UIMA

2.

Column A	Column B
JSON	SOAP
MongoDB	REST
XML	JSON
Flexible structure	CouchDB
JSON	XML

Answer:

Column A	Column B
JSON	REST
MongoDB	JSON
XML	SOAP
Flexible structure	XML
JSON	CouchDB

C. Solve Me

You are a senior faculty at a premier engineering institute of the city. The Head of the Department has asked you to take a look at the institute's learning website and make a list of the unstructured data that gets generated on the website that can then be stored and analyzed to improve the website to facilitate and enhance the student's learning. You log into the institute's learning website and observe the following features on it:

- Presentation decks (.pdf files)
- Laboratory Manual (.doc files)
- Discussion forum
- Student's blog
- Link to Wikipedia
- A survey questionnaire for the students
- Student's performance sheet downloadable into an .xls sheet
- Student's performance sheet downloadable into a .txt file
- Audio/Video learning files (.wav files)
- .xls sheet having a compiled list of FAQs

From this list, you select the following as sources of unstructured data:

1.
2.
3.
4.
5.
6.

You have just finished making your list when your colleague comes in looking for you. Both of you decide to go away to the cafeteria in the vicinity of the institute's campus. You have forever liked this cafeteria. And you have reasons for the same. There are a couple of machines in the cafeteria's reception area that the customers can use to feed in their orders from a selection of menu items. Once the order is done, you are given a token number. Once your order is ready for serving, the display flashes your token number. It goes without saying

that the billing is also automated. You being in the IT department cannot refrain from thinking about the data that gets collected by these automatic applications. Here's your list:

1.
2.
3.
4.
5.
6.

You are thinking of the analysis that you can perform on this data. Here's your list:

1.
2.
3.
4.
5.
6.

D. Solved Exercises

1. Why is an email placed in the "unstructured category"?

   Answer: Let us take a look at what we can place in the body of the email. We can have any or more of the following:

   - Hyperlink
   - PDFs/DOCs/XLS/etc. attachments
   - Emoticons
   - Images
   - Audio/video attachments
   - Free flowing text, etc.

   The above are reasons behind placing the email in the "unstructured category".

2. What category will you place a CCTV footage into?

   Answer: Unstructured

3. You have just got a book issued from the library. What are the details about the book that can be placed in an RDBMS table?

   Answer:

   - Title of the book
   - Author of the book
   - Publisher of the book
   - Year of Publication
   - No. of pages in the book
   - Type of book such as whether hardbound or paperback
   - Price of the book
   - ISBN No. of the book
   - Attachments such as With CD or Without CD, etc.

4. Which category would you place the consumer complaints and feedback?
   Answer: Unstructured data

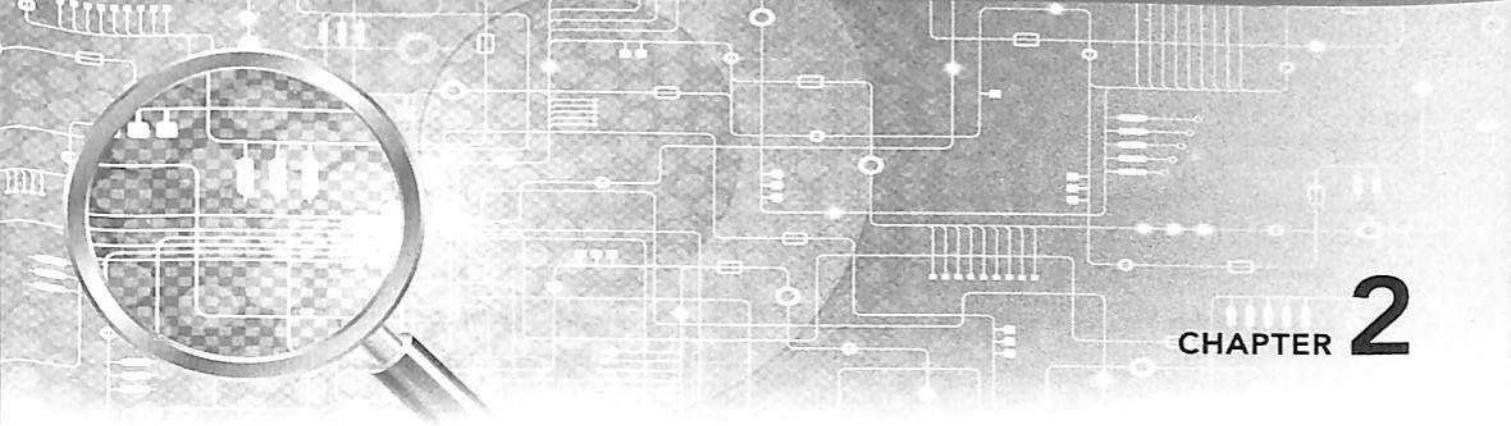
E. Unsolved Exercises

1. Which category (structured, semi-structured, or unstructured) will you place a web page in?
2. What according to you are the challenges with unstructured data?
3. Which category (structured, semi-structured, or unstructured) will you place a PowerPoint presentation in?
4. Which category (structured, semi-structured, or unstructured) will you place a Word Document in?
5. State a few examples of human generated and machine-generated data.

SCENARIO-BASED QUESTION

You are at the university library. You see a few students browsing through the library catalog on a kiosk. You observe the librarians busy at work issuing and returning books. You see a few students fill up the feedback form on the services offered by the library. Quite a few students are learning using the e-learning content.

Think for a while on the different types of data that are being generated in this scenario. Support your answer with logic.



CHAPTER 2

Introduction to Big Data

BRIEF CONTENTS

"Data is the new science. Big Data holds the answers."

– Pat Gelsinger, the Chief Executive Officer of VMware, Inc. and former Chief Operating Officer of EMC Corporation

WHAT'S IN STORE?

This chapter focuses on defining and explaining big data. The "Internet of Things" and its widely ultra-connected nature are leading to a burgeoning rise in big data. There is no dearth of data for today's enterprise. On the contrary, they are mired in data and quite deep at that. That brings us to the following questions:

1. Why is it that we cannot forego big data?
2. How has it come to assume such magnanimous importance in running business?

3. How does it compare with the traditional Business Intelligence (BI) environment?
4. Is it here to replace the traditional, relational database management system and data warehouse environment or is it likely to complement their existence?"

Data is widely available. What is scarce is the ability to extract wisdom from it.

Hal Varian, Google's Chief Economist, 2010

PICTURE THIS...

You recently availed the opportunity to attend a virtual classroom session from a leading training institute. You are reflecting back on the experience. Since the session was on big data, it gets you thinking on the types and volume of data that was created before, during, and after the session. It all began with you registering online a week ago for the "Big Data" course. You remember having received an acknowledgment confirming your registration. They had also stated that they will send across some reading contents two days prior to the session. And true to their word, they did. When you logged into the session, you saw that there were 493 other participants. The presenter was introducing the process on smooth learning through the session. During the session, the participants could converse with the presenter as well as with other participants using the chat facility. They had also activated a discussion forum for participants to share their learnings/views/opinions/experiences, etc. There were assignments, which would have to be attempted and submitted on their site. There was an assessment towards the end of the session that was graded. There was a feedback form that was made available at the end of the session to hear back from the participants. They also provided additional reading contents in the form of references to white papers/research papers. The lecture was recorded and made available for better learning and comprehension of the participants.

It was a good experience and you are already thinking of being part of another such experience very soon.

There is no dearth of such virtual classroom sessions being conducted today. There is a huge learning community out there eager to learn. Just think on the volume of data that gets generated, and the variety (the list of attendees, their scores and grades, their chat conversations, their assignments, the polling questions put forth by the instructor to gauge the level of understanding and participation from the learners, etc.) of data that we produce as well consume as we become part of these virtual training sessions.

2.1 CHARACTERISTICS OF DATA

Let us start with the characteristics of data. As depicted in Figure 2.1, data has three key characteristics:

1. **Composition:** The composition of data deals with the structure of data, that is, the sources of data, the granularity, the types, and the nature of data as to whether it is static or real-time streaming.
2. **Condition:** The condition of data deals with the state of data, that is, "Can one use this data as is for analysis?" or "Does it require cleansing for further enhancement and enrichment?"
3. **Context:** The context of data deals with "Where has this data been generated?" "Why was this data generated?" "How sensitive is this data?" "What are the events associated with this data?" and so on.

Small data (data as it existed prior to the big data revolution) is about certainty. It is about fairly known data sources; it is about no major changes to the composition or context of data.

Most often we have answers to queries like why this data was generated, where and when it was generated, exactly how we would like to use it, what questions will this data be able to answer, and so on. Big data is

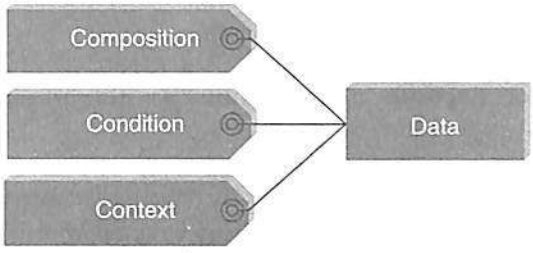


Figure 2.1 Characteristics of data.

about complexity... complexity in terms of multiple and unknown datasets, in terms of exploding volume, in terms of the speed at which the data is being generated and the speed at which it needs to be processed, and in terms of the variety of data (internal or external, behavioral or social) that is being generated.

2.2 EVOLUTION OF BIG DATA

1970s and before was the era of mainframes. The data was essentially primitive and structured. Relational databases evolved in 1980s and 1990s. The era was of data intensive applications. The World Wide Web (WWW) and the Internet of Things (IoT) have led to an onslaught of structured, unstructured, and multimedia data. Refer Table 2.1.

Table 2.1 The evolution of big data

	Data Generation and Storage	Data Utilization	Data Driven
Complex and Unstructured			Structured data, unstructured data, multimedia data
Complex and Relational		Relational databases: Data-intensive applications	
Primitive and Structured	Mainframes: Basic data storage		
	1970s and before	**Relational (1980s and 1990s)**	**2000s and beyond**

2.3 DEFINITION OF BIG DATA

If we were to ask you the simple question: "Define Big Data", what would your answer be? Well, we will give you a few responses that we have heard over time:

1. Anything beyond the human and technical infrastructure needed to support storage, processing, and analysis.
2. Today's BIG may be tomorrow's NORMAL.

3. Terabytes or petabytes or zettabytes of data.
4. I think it is about 3 Vs.

Refer Figure 2.2. Well, all of these responses are correct. But it is not just one of these; in fact, big data is all of the above and more.

> *Big data is high-volume, high-velocity, and high-variety information assets that demand cost effective, innovative forms of information processing for enhanced insight and decision making.*
>
> *Source:* Gartner IT Glossary

The 3Vs concept was proposed by the Gartner analyst Doug Laney in a 2001 MetaGroup research publication, titled, *3D Data Management: Controlling Data Volume, Variety and Velocity.*

Source: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf

For the sake of easy comprehension, we will look at the definition in three parts. Refer Figure 2.3.

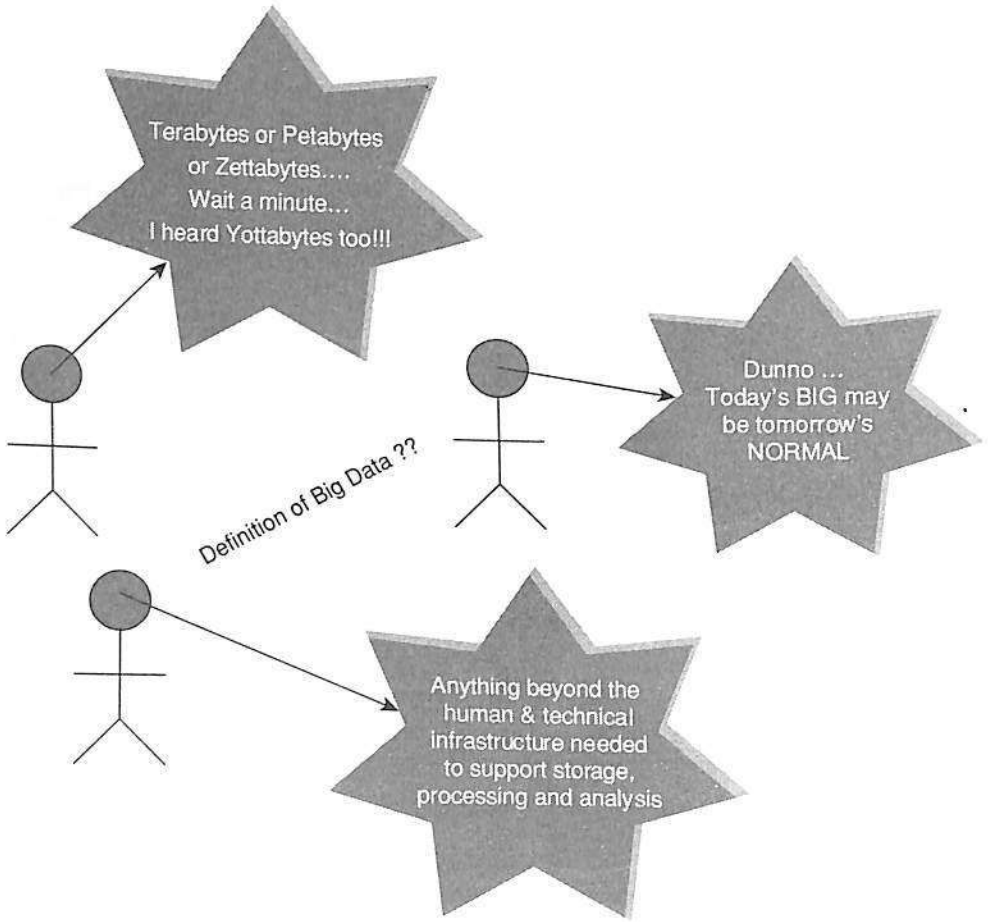


Figure 2.2 Definition of big data.

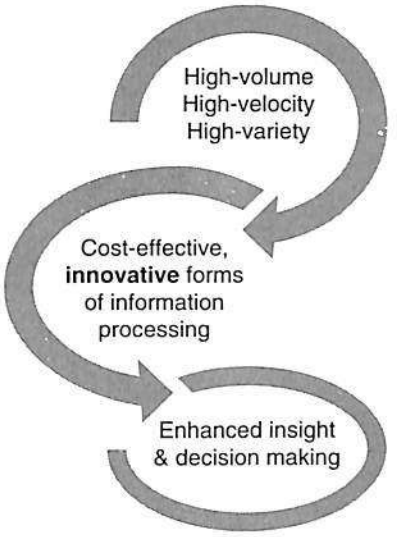


Figure 2.3 Definition of big data – Gartner.

Part I of the definition "big data is high-volume, high-velocity, and high-variety information assets" talks about voluminous data (humongous data) that may have great variety (a good mix of structured, semi-structured, and unstructured data) and will require a good speed/pace for storage, preparation, processing, and analysis.

Part II of the definition "cost effective, innovative forms of information processing" talks about embracing new techniques and technologies to capture (ingest), store, process, persist, integrate, and visualize the high-volume, high-velocity, and high-variety data.

Part III of the definition "enhanced insight and decision making" talks about deriving deeper, richer, and meaningful insights and then using these insights to make faster and better decisions to gain business value and thus a competitive edge.

Data → Information → Actionable intelligence → Better decisions → Enhanced business value

2.4 CHALLENGES WITH BIG DATA

Refer Figure 2.4. Following are a few challenges with big data:

1. Data today is growing at an exponential rate. Most of the data that we have today has been generated in the last 2–3 years. This high tide of data will continue to rise incessantly. The key questions here are: "Will all this data be useful for analysis?", "Do we work with all this data or a subset of it?", "How will we separate the knowledge from the noise?", etc.
2. Cloud computing and virtualization are here to stay. Cloud computing is the answer to managing infrastructure for big data as far as cost-efficiency, elasticity, and easy upgrading/downgrading is concerned. This further complicates the decision to host big data solutions outside the enterprise.
3. The other challenge is to decide on the period of retention of big data. Just how long should one retain this data? A tricky question indeed as some data is useful for making long-term decisions, whereas in few cases, the data may quickly become irrelevant and obsolete just a few hours after having being generated.

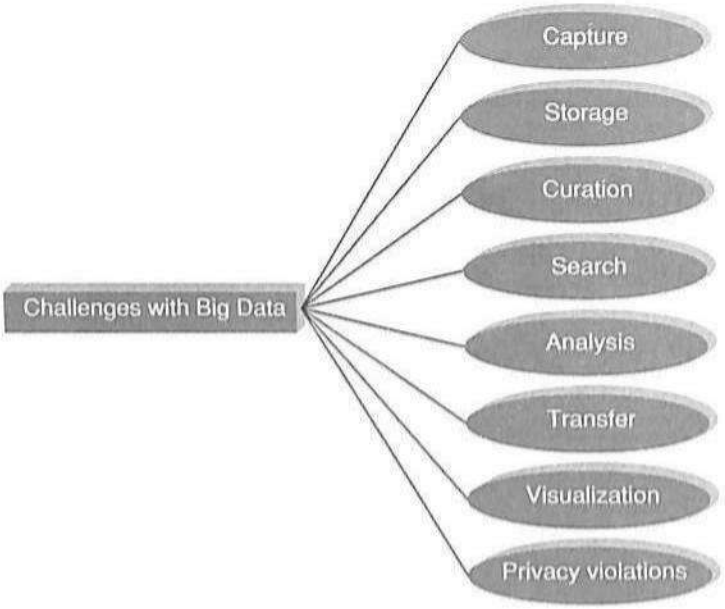


Figure 2.4 Challenges with big data.

4. There is a dearth of skilled professionals who possess a high level of proficiency in data sciences that is vital in implementing big data solutions.
5. Then, of course, there are other challenges with respect to capture, storage, preparation, search, analysis, transfer, security, and visualization of big data. Big data refers to datasets whose size is typically beyond the storage capacity of traditional database software tools. There is no explicit definition of how big the dataset should be for it to be considered "big data." Here we are to deal with data that is just too big, moves way to fast, and does not fit the structures of typical database systems. The data changes are highly dynamic and therefore there is a need to ingest this as quickly as possible.
6. Data visualization is becoming popular as a separate discipline. We are short by quite a number, as far as business visualization experts are concerned.

2.5 WHAT IS BIG DATA?

Big data is data that is big in volume, velocity, and variety. Refer Figure 2.5.

2.5.1 Volume

We have seen it grow from bits to bytes to petabytes and exabytes. Refer Table 2.2 and Figure 2.6.

Bits → Bytes → Kilobytes → Megabytes → Gigabytes → Terabytes → Petabytes → Exabytes → Zettabytes → Yottabytes

2.5.1.1 Where Does This Data get Generated?

There are a multitude of sources for big data. An XLS, a DOC, a PDF, etc. is unstructured data; a video on YouTube, a chat conversation on Internet Messenger, a customer feedback form on an online retail website

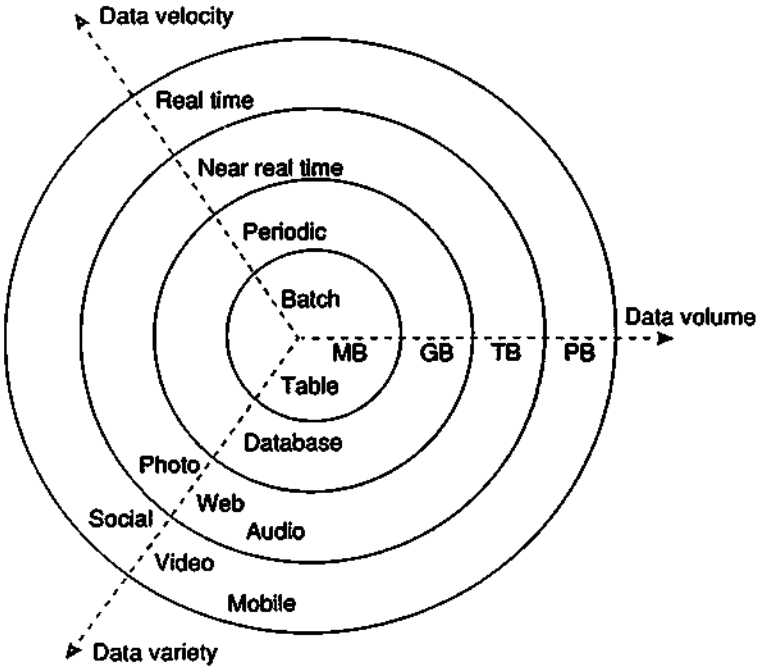


Figure 2.5 Data: Big in volume, variety, and velocity.

Table 2.2 Growth of data

Bits	0 or 1
Bytes	8 bits
Kilobytes	1024 bytes
Megabytes	1024^2 bytes
Gigabytes	1024^3 bytes
Terabytes	1024^4 bytes
Petabytes	1024^5 bytes
Exabytes	1024^6 bytes
Zettabytes	1024^7 bytes
Yottabytes	1024^8 bytes

is unstructured data; a CCTV coverage, a weather forecast report is unstructured data too. Refer Figure 2.7 for the sources of big data.

1. **Typical internal data sources:** Data present within an organization's firewall. It is as follows:
   - ***Data storage:*** File systems, SQL (RDBMSs – Oracle, MS SQL Server, DB2, MySQL, PostgreSQL, etc.), NoSQL (MongoDB, Cassandra, etc.), and so on.
   - ***Archives:*** Archives of scanned documents, paper archives, customer correspondence records, patients' health records, students' admission records, students' assessment records, and so on.

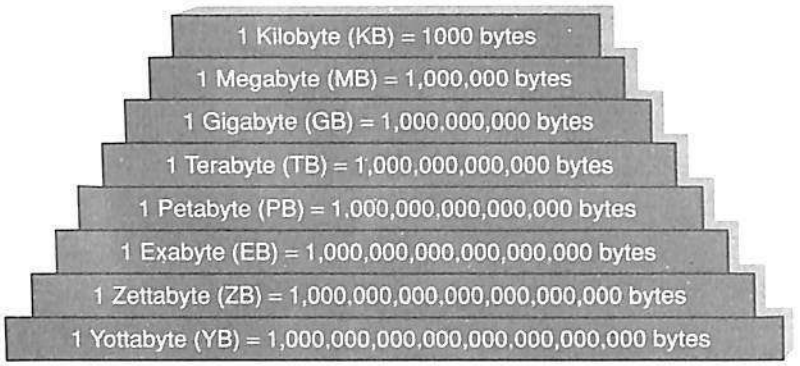


Figure 2.6 A mountain of data.

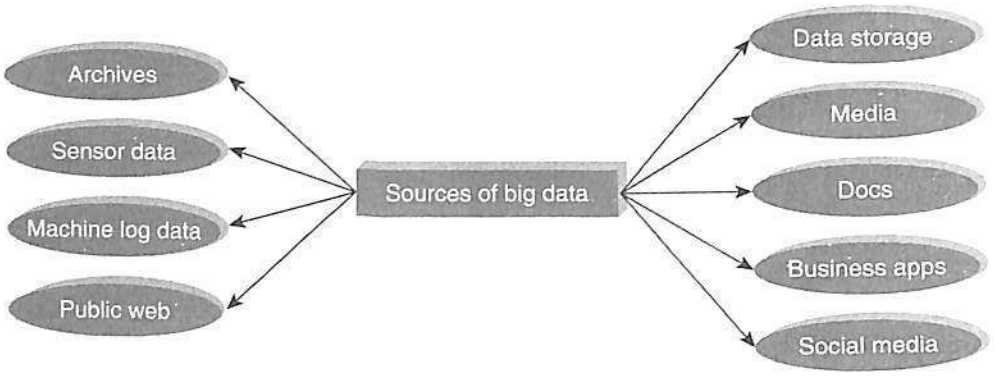


Figure 2.7 Sources of big data.

2. **External data sources:** Data residing outside an organization's firewall. It is as follows:
   - ***Public Web:*** Wikipedia, weather, regulatory, compliance, census, etc.
3. **Both (internal + external data sources)**
   - ***Sensor data:*** Car sensors, smart electric meters, office buildings, air conditioning units, refrigerators, and so on.
   - ***Machine log data:*** Event logs, application logs, Business process logs, audit logs, clickstream data, etc.
   - ***Social media:*** Twitter, blogs, Facebook, LinkedIn, YouTube, Instagram, etc.
   - ***Business apps:*** ERP, CRM, HR, Google Docs, and so on.
   - ***Media:*** Audio, Video, Image, Podcast, etc.
   - ***Docs:*** Comma separated value (CSV), Word Documents, PDF, XLS, PPT, and so on.

2.5.2 Velocity

We have moved from the days of batch processing (remember our payroll applications) to real-time processing.

Batch → Periodic → Near real time → Real-time processing

2.5.3 Variety

Variety deals with a wide range of data types and sources of data. We will study this under three categories: Structured data, semi-structured data and unstructured data.

1. **Structured data:** From traditional transaction processing systems and RDBMS, etc.
2. **Semi-structured data:** For example Hyper Text Markup Language (HTML), eXtensible Markup Language (XML).
3. **Unstructured data:** For example unstructured text documents, audios, videos, emails, photos, PDFs, social media, etc.

2.6 OTHER CHARACTERISTICS OF DATA WHICH ARE NOT DEFINITIONAL TRAITS OF BIG DATA

There are yet other characteristics of data which are not necessarily the definitional traits of big data. Few of these are listed as follows:

1. **Veracity and validity:** *Veracity* refers to biases, noise, and abnormality in data. The key question here is: "Is all the data that is being stored, mined, and analyzed meaningful and pertinent to the problem under consideration?" *Validity* refers to the accuracy and correctness of the data. Any data that is picked up for analysis needs to be accurate. It is not just true about big data alone.
2. **Volatility:** Volatility of data deals with, how long is the data valid? And how long should it be stored? There is some data that is required for long-term decisions and remains valid for longer periods of time. However, there are also pieces of data that quickly become obsolete minutes after their generation.
3. **Variability:** Data flows can be highly inconsistent with periodic peaks.

PICTURE THIS...

An online retailer announces the "big sale day" for a particular week. The retailer is likely to experience an upsurge in customer traffic to the website during this week. In the same way, he/she might experience a slump in his/her business immediately after the festival season. This reemphasizes the point that one might witness spikes in data at some point in time and at other times, the data flow can go flat.

2.7 WHY BIG DATA?

The more data we have for analysis, the greater will be the analytical accuracy and also the greater would be the confidence in our decisions based on these analytical findings. This will entail a greater positive impact in terms of enhancing operational efficiencies, reducing cost and time, and innovating on new products, new services, and optimizing existing services. Refer Figure 2.8.

More data → More accurate analysis → Greater confidence in decision making → Greater operational efficiencies, cost reduction, time reduction, new product development, and optimized offerings, etc.

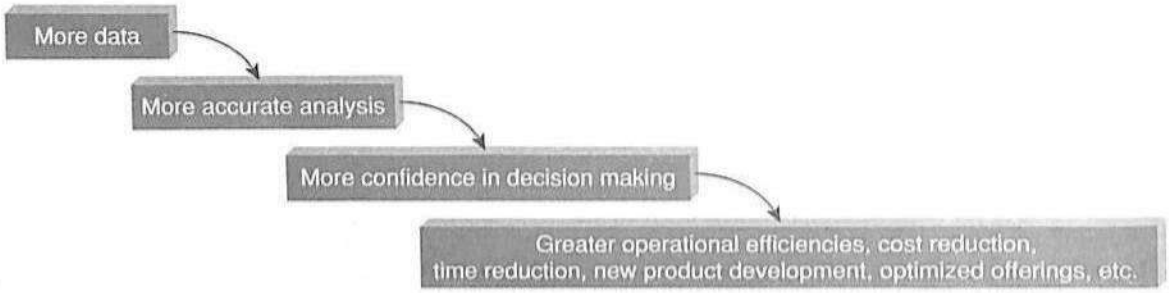


Figure 2.8 Why big data?

2.8 ARE WE JUST AN INFORMATION CONSUMER OR DO WE ALSO PRODUCE INFORMATION?

PICTURE THIS...

You have been invited to your friend's promotion party. You are happy and excited to join your friend at this important milestone in her career. You send in your confirmation through a text message. You get ready and leave for your friend's residence. On the way, you stop at a gas station to refuel. You pay using your credit card. You stop at an upmarket Archie's store to pick a good greeting card and a gift. You get the items billed at the Point of Sale system and pay cash at the counter. While at the party, you click photographs and post it on Facebook, Flickr, and the likes. Within minutes, you start to get likes and comments on your posts.

Mention the places in this scenario where data was generated:

1. Text message to send in the confirmation to attend the promotion bash.
2. Use of credit card to pay for gas/fuel at the gas station.
3. Point of Sale system at Archie's where your transaction gets recorded.
4. Photographs and posts on social networking sites.
5. Likes and comments to your post.

Likewise, there are several instances everyday where you generate data. Think about cases where you are a consumer of information.

2.9 TRADITIONAL BUSINESS INTELLIGENCE (BI) VERSUS BIG DATA

Let us take a sneak peek into some of the differences that one encounters dealing with traditional BI and big data.

1. In traditional BI environment, all the enterprise's data is housed in a central server whereas in a big data environment data resides in a distributed file system. The distributed file system scales by scaling in or out horizontally as compared to typical database server that scales vertically.
2. In traditional BI, data is generally analyzed in an offline mode whereas in big data, it is analyzed in both real time as well as in offline mode.

3. Traditional BI is about structured data and it is here that data is taken to processing functions (move data to code) whereas big data is about variety: Structured, semi-structured, and unstructured data and here the processing functions are taken to the data (move code to data).

2.10 A TYPICAL DATA WAREHOUSE ENVIRONMENT

Let us look at a typical Data Warehouse (DW) environment. Operational or transactional or day-to-day business data is gathered from Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM), legacy systems, and several third party applications. The data from these sources may differ in format [data could have been housed in any RDBMS such as Oracle, MS SQL Server, DB2, MySQL, and Teradata, and so on or in spreadsheet (.xls, .xlsx, etc.) or .csv or txt]. Data may come from data sources located in the same geography or different geographies. This data is then integrated, cleaned up, transformed, and standardized through the process of Extraction, Transformation, and Loading (ETL). The transformed data is then loaded into the enterprise data warehouse (available at the enterprise level) or data marts (available at the business unit/ functional unit or business process level). A host of market leading business intelligence and analytics tools are then used to enable decision making from the use of ad-hoc queries, SQL, enterprise dashboards, data mining, etc. Refer Figure 2.9.

2.11 A TYPICAL HADOOP ENVIRONMENT

Let us now study the Hadoop environment. Is it very different from the data warehouse environment and where exactly is this difference?

As is fairly obvious from Figure 2.10, the data sources are quite disparate from web logs to images, audios, and videos to social media data to the various docs, pdfs, etc. Here the data in focus is not just the data within the company's firewall but also data residing outside the company's firewall. This data is placed in Hadoop Distributed File System (HDFS). If need be, this can be repopulated back to operational systems or fed to the enterprise data warehouse or data marts or Operational Data Store (ODS) to be picked for further processing and analysis.

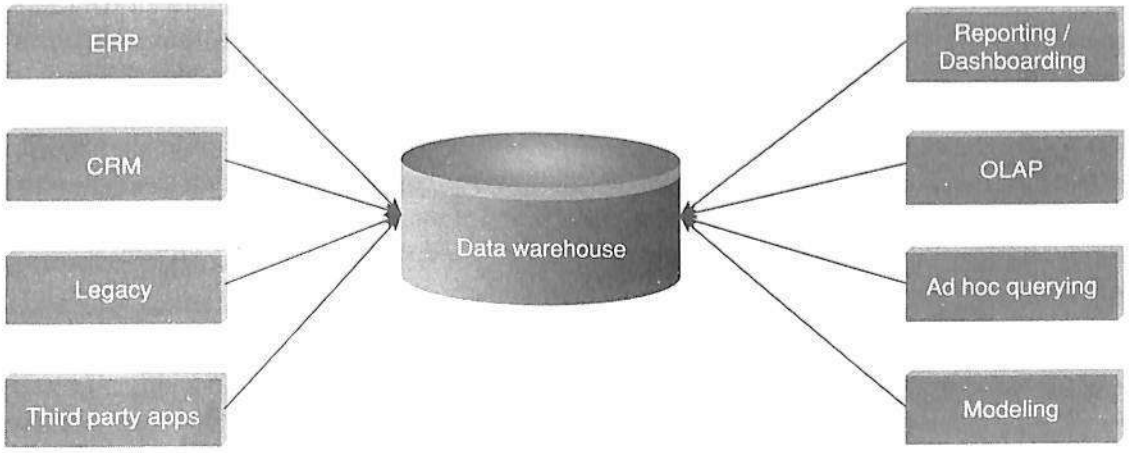


Figure 2.9 A typical data warehouse environment.

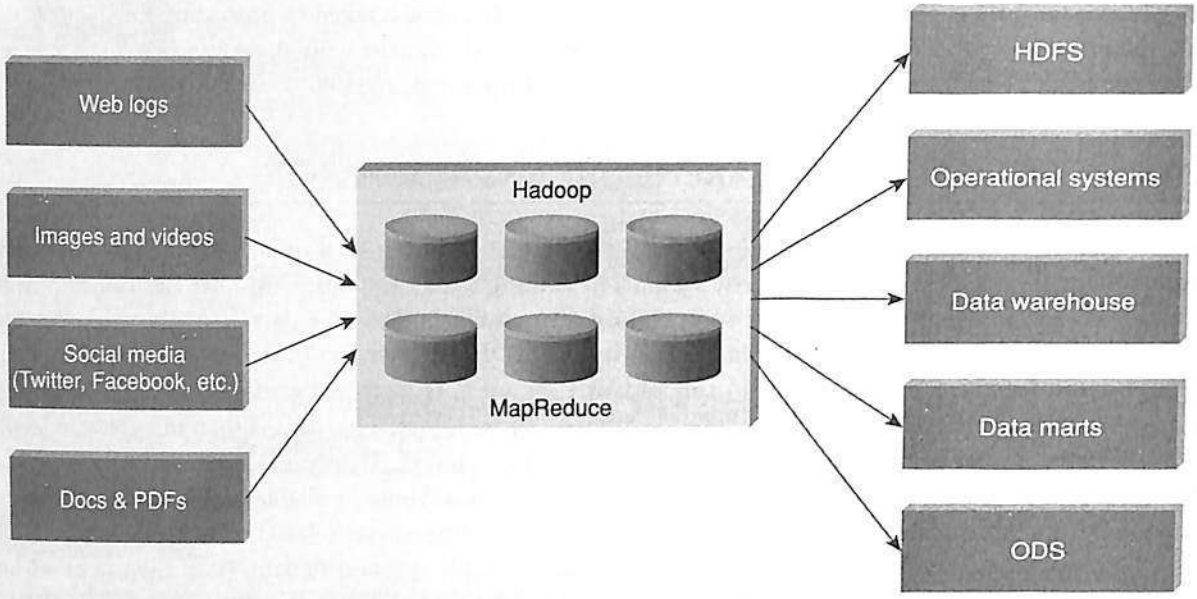


Figure 2.10 A typical Hadoop environment.

2.12 WHAT IS NEW TODAY?

A coexistence strategy that combines the best of legacy data warehouse and analytics environment with the new power of big data solutions is the best of both the worlds. Refer Figure 2.11.

2.12.1 Coexistence of Big Data and Data Warehouse

It is NOT about rip and replace. It will not be possible to get rid of RDBMS or massively parallel processing (MPP), but instead use the right tool for the right job.

As we are aware that few companies are a wee bit comfortable working with incumbent data warehouse for standard BI and analytics reporting, for example the quarterly sales report, customer dashboard, etc. The data warehouse can continue with its standard workload drawing data from legacy operational systems, storing the historical data to provision traditional BI reporting and analytics needs. However, one will not be able to ignore the power that Hadoop brings to the table with different types of analysis on different types of data. The same operational systems, which till now was engaged in powering the data warehouse, can also populate the big data environment when they're needed for computation-rich processing or for raw data exploration. It will be a tight balancing act to steer the workload to the right platform based on what that platform was designed to do.

Here is a thought-provoking piece from Ralph Kimball at a cloudera webinar:

> *"Here's a question that made me laugh a little bit, but it's a serious question: 'Well does this mean that relational databases are going to die?'. I think that there was a sense, three or four years ago, that maybe this was all a giant zero sum game between Hadoop and relational databases, and that has*

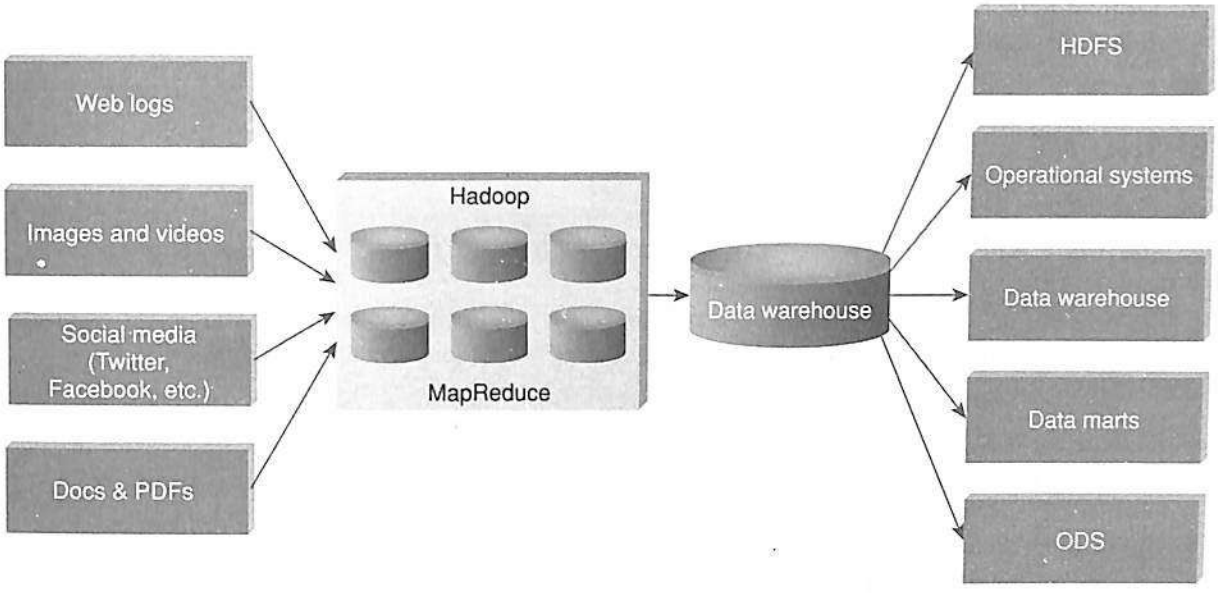


Figure 2.11 Big data and data warehouse coexistence.

> *simply gone away. Everyone has now realized that there's a huge legacy value in relational databases for the purposes they are used for. Not only transaction processing, but for all the much focused, index-oriented queries on that kind of data, and that will continue in a very robust way forever. Hadoop, therefore, will present this alternative kind of environment for different types of analysis for different kinds of data, and the two of them will coexist. And they will call each other. There may be points at which the business user isn't actually quite sure which one of them they are touching at any point of time."*

Just as one cannot ignore the powerful analytics capability of Hadoop, one will not be able to ignore the revolutionary developments in RDBMS such as in-memory processing, etc. The need of the hour is to have both data warehouse and Hadoop co-exist in today's environment.

2.13 WHAT IS CHANGING IN THE REALMS OF BIG DATA?

Gone are the days when IT and business could work in silos and still see the business through. Today, it is an era of a tight handshake between business, IT, and yet another class called *Data Scientists* (more on it in Chapter 3 on "*Big Data Analytics*"). We are citing three very important reasons why companies should compulsorily consider leveraging big data:

1. **Competitive advantage:** The most important resource with any organization today is their data. What they do with it will determine their fate in the market.
2. **Decision making:** Decision making has shifted from the hands of the elite few to the empowered many. Good decisions play a significant role in furthering customer engagement, reducing operating margins in retail, cutting cost and other expenditures in the health sector.

3. **Value of data:** The value of data continues to see a steep rise. As the all-important resource, it is time to look at newer architecture, tools, and practices to leverage this.

REMIND ME

- The World Wide Web (WWW) and the Internet of Things (IoT) have led to an onslaught of structured, unstructured, and multimedia data.
- *Big data is high-volume, high-velocity, and high-variety information assets that demand cost effective, innovative forms of information processing for enhanced insight and decision making.*

  Source: Gartner IT Glossary
- More data → More accurate analysis → Greater confidence in decision making → Greater operational efficiencies, cost reduction, time reduction, new product development, and optimized offerings, etc.
- Traditional BI is about structured data and it is here that data is taken to processing functions (move data to code). On the other hand, big data is about variety: Structured, semi-structured, and unstructured data and here the processing functions are taken to the data (move code to data).

POINT ME (BOOK)

- Big Data for Dummies – Judith Hurwitz, Alan Nugent, Fern Halper, Marcia Kaufman, Wiley India Pvt. Ltd.

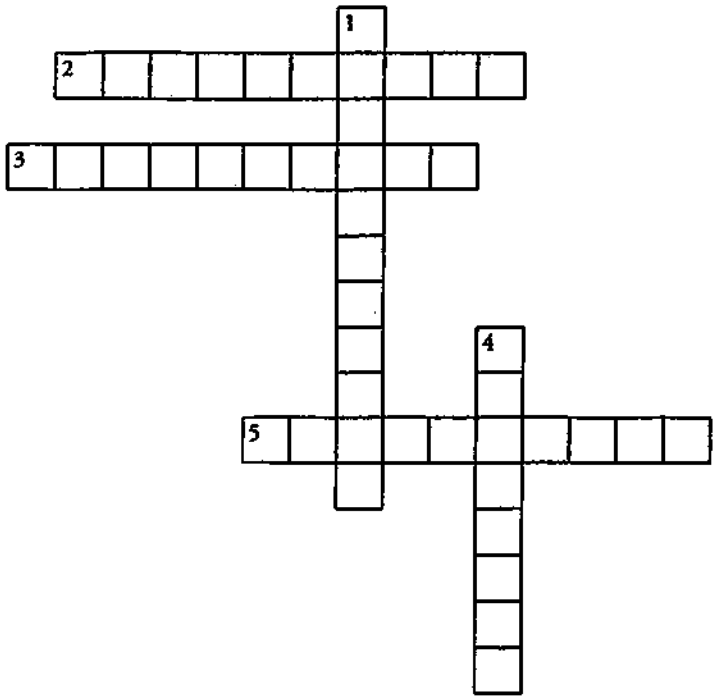
CONNECT ME (INTERNET RESOURCES)

- http://en.wikipedia.org/wiki/Big_data
- http://www.sas.com/en_us/insights/big-data/what-is-big-data.html
- https://www.oracle.com/bigdata/
- http://bigdatauniversity.com/
- http://www.sap.com/solution/big-data/software/overview.html
- http://www.ibm.com/software/data/bigdata/
- http://www.ibm.com/big-data/us/en/
- http://www.sas.com/en_us/insights/big-data/what-is-big-data.html
- http://timoelliott.com/blog/2014/04/no-hadoop-isnt-going-to-replace-your-data-warehouse.html

TEST ME

A. Crossword

Puzzle on Big Data



Across

2. _____, a Gartner analyst coined the term, 'Big Data'
3. _____, is the characteristic of data dealing with its retention.
5. _____, is a large data repository that stores data in its native format until it is needed.

Down

1. _____ characteristic of data explains the spikes in data.
4. Near real time processing or real time processing deals with _____ characteristic of data.

Answer:

Across

2. Doug Laney
3. Volatility
5. Data Lakes

Down

1. Variability
4. Velocity

B. Fill Me

1. Big data is high-volume, high-velocity, and high-variety information assets that demand _____ _____ forms of information processing for enhanced _____ and _____.

Answer: Cost-effective, Innovative, Insight, Decision making

C. Match the Following

Column A	Column B
PostgreSql	Machine generated unstructured data
Scientific data	Open source relational database
Point-of-sale	Human-generated unstructured data
Social Media data	Machine-generated structured data
Gaming-related data	Human-generated unstructured data
Mobile data	Human-generated structured data

Answer:

Column A	Column B
PostgreSql	Open source relational database
Scientific data	Machine generated unstructured data
Point-of-sale	Machine-generated structured data
Social Media data	Human-generated unstructured data
Gaming-related data	Human-generated structured data
Mobile data	Human-generated unstructured data

D. Unsolved Exercises

1. Share your understanding of big data.
2. How is traditional BI environment different from the big data environment?
3. *Big data (Hadoop) will replace the traditional RDBMS and data warehouse.* Comment.
4. Share your experience as a customer on an e-commerce site. Comment on the big data that gets created on a typical e-commerce site.
5. What is your understanding of "Big Data Analytics"?

CHALLENGE ME

1. What is Internet of Things and why does it matter?

   Answer: See http://www.sas.com/en_us/insights/big-data/internet-of-things.html

2. Can the same visualization tool that we run over conventional data warehouse, be used in big data environment?

   Answer: Let us look at Figure 2.12 to understand the solution:

   As per Figure 2.12, structured data is stored in Relational Database Management System (RDBMS) whereas big data (largely unstructured data) is stored in NoSQL databases. Structured data after cleansing, transforming, and converting to a uniform standard format are placed in the enterprise data warehouse

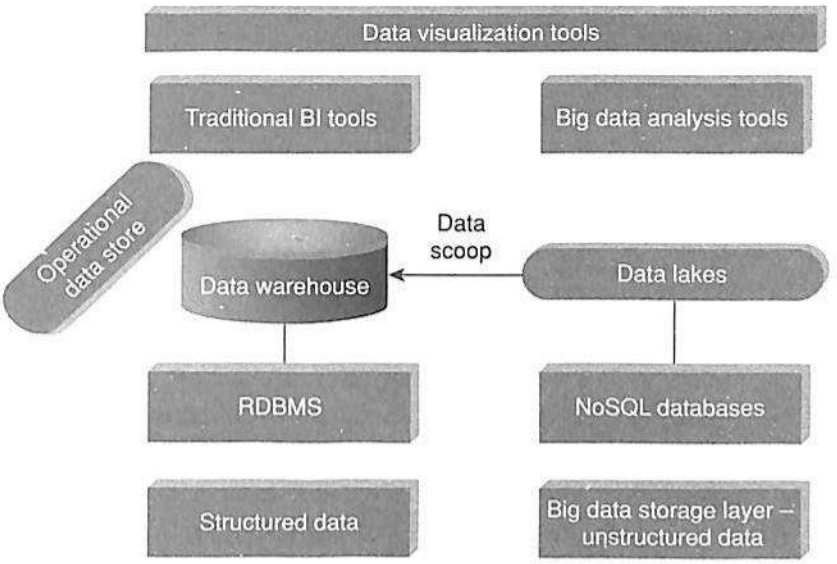


Figure 2.12 Visualization tools for traditional BI and big data.

(at the enterprise level) or the data marts (at the business unit or function level) or operational data stores (almost the complete operational data of an enterprise is housed here) whereas the good variety of data (structured, semi-structured, and unstructured data) is placed in data lakes (a large data repository that stores raw data in its native format until it is needed). Data can then be scooped from data lakes to data warehouses and traditional BI tools can then be run over them. A common set of data visualization tools can then be used to present results after analysis. This goes to emphasize the point, that it makes sense to use the tool that is a specialist for a particular function for example RDBMS for structured data and NoSQL for voluminous data that may be schema less.

CHAPTER 3

Big Data Analytics

BRIEF CONTENTS

- What's in Store?
- Where do we Begin?
- What is Big Data Analytics?
- What Big Data Analytics isn't?
- Why this Sudden Hype Around Big Data Analytics?
- Classification of Analytics
- Greatest Challenges that Prevent Businesses from Capitalizing on Big Data
- Top Challenges Facing Big Data
- Why is Big Data Analytics Important?
- What Kind of Technologies are we Looking Toward to Help Meet the Challenges Posed by Big Data?
- Data Science
- Data Scientist … Your New Best Friend!!!
- Terminologies Used in Big Data Environment
  - In-Memory Analytics
  - In-Database Processing
  - Symmetric Multiprocessor System
  - Massively Parallel Processing
  - Difference between Parallel and Distributed Systems
  - Shared Nothing Architecture
  - Consistency, Availability, Partition Tolerance (CAP) Theorem Explained
  - Basically Available Soft State Eventual Consistency (BASE)
- Few Top Analytics Tools

"If you do not know how to ask the right question, you discover nothing."

– W. Edwards Deming

WHAT'S IN STORE?

This chapter is about understanding "Big Data Analytics." We have taken you through the comprehension of the term Big Data – datasets which are voluminous, rich in variety, and calls for processing at a great speed. Big data analytics is the process of examining these large datasets of big data – to unearth hidden

patterns, decipher unknown correlations, understand the rationale behind market trends, and recognize customer preferences and other useful business information. The analytical findings can lead to more effective marketing, better customer service and satisfaction, newer products and services, improved operational efficiency, reduced expenditure, competitive advantages over rival organizations, boosted business gains, etc.

PICTURE THIS...

Scenario 1: You have heard a lot from your friends about the deals on offer on the Amazon site. You decide to register on www.amazon.co.in to avail their discount offers and bumper sales.

A couple of days later, you make a purchase on their site. You landed yourself a good deal by going for books by your favorite author. There is something that does not escape your attention. Amazon has made a few suggestions (of books on similar topics or books by the same author) to you to help with your next or future purchases. You wonder how Amazon's recommendation engine was able to do this for you. Is it something that they do for all their customers?

Well, Amazon's recommendation engine churns out these sort of good suggestions for customers like you, day in and day out. The company gathers all information about your past purchases together with what it knows about you, studies your buying patterns, and the buying patterns of customers like you to arrive at the recommendations that can help with your future purchase. At the core they have big data analytics working for them.

Scenario 2: You are the owner of a trucks transport company. Your company has 500 trucks plying several routes and carrying cargo from one place to another. It is one of those busy days where almost all the trucks are engaged in carrying cargo. You get a call to help with a cargo delivery. They are ready to pay double the charge. You do not want to miss this opportunity. But which truck should you engage. The one that is the nearest but is facing the heaviest traffic or the second nearest one but that is occupied to 75% and will not be able to take more load. There is a need to analyze the truck load, the fuel consumption, the traffic on various routes, etc. before deciding on which truck to select to pick up the new delivery.

3.1 WHERE DO WE BEGIN?

Raw data is collected, classified, and organized. Associating it with adequate metadata and laying bare the context converts this data into meaningful information. It is then aggregated and summarized so that it becomes easy to consume it for analysis. Gradual accumulation of such meaningful information builds a knowledge repository. This, in turn, helps with actionable insights which prove useful for decision making. Refer Figure 3.1.

Organizations have realized that they will not be able to ignore big data if they want to be competitive enough and make those timely decisions to make well of the fleeting opportunities. They will have to analyze

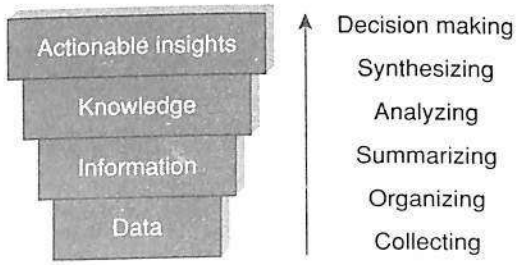


Figure 3.1 Transformation of data to yield actionable insights.

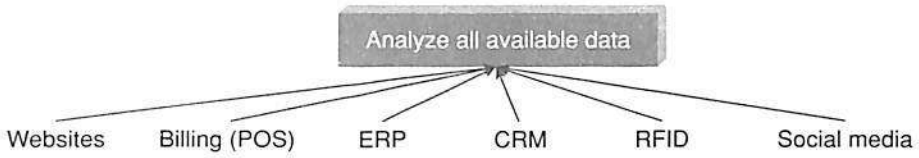


Figure 3.2 Types of unstructured data available for analysis.

big time and also take into consideration big data that makes it to the organization at unprecedented level in terms of volume, velocity, and variety.

Big data analytics is the process of examining big data to uncover patterns, unearth trends, and find unknown correlations and other useful information to make faster and better decisions. Analytics begin with analyzing all available data. Refer Figure 3.2.

3.2 WHAT IS BIG DATA ANALYTICS?

Big Data Analytics is...

1. **Technology-enabled analytics:** Quite a few data analytics and visualization tools are available in the market today from leading vendors such as IBM, Tableau, SAS, R Analytics, Statistica, World Programming Systems (WPS), etc. to help process and analyze your big data.
2. About gaining a meaningful, deeper, and richer insight into your business to steer it in the right direction, understanding the customer's demographics to cross-sell and up-sell to them, better leveraging the services of your vendors and suppliers, etc.

   Author's experience: The other day I was pleasantly surprised to get a few recommendations via email from one of my frequently visited online retailers. They had recommended clothing line from my favorite brand and also the color suggested was one to my liking. How did they arrive at this? In the recent past, I had been buying clothing line of a particular brand and the color preference was pastel shades. They had it stored in their database and pulled it out while making recommendations to me.
3. About a competitive edge over your competitors by enabling you with findings that allow quicker and better decision-making.
4. A tight handshake between three communities: IT, business users, and data scientists. Refer Figure 3.3.
5. Working with datasets whose volume and variety exceed the current storage and processing capabilities and infrastructure of your enterprise.
6. About moving code to data. This makes perfect sense as the program for distributed processing is tiny (just a few KBs) compared to the data (Terabytes or Petabytes today and likely to be Exabytes or Zettabytes in the near future).

3.3 WHAT BIG DATA ANALYTICS ISN'T?

We have often asked participants of our learning programs as what comes to mind when you hear the term "Big Data." And we are not surprised by the answer... it is "Volume." But now that we have a clear understanding of big data, we know it isn't only about volume but the variety and velocity too are very important factors.

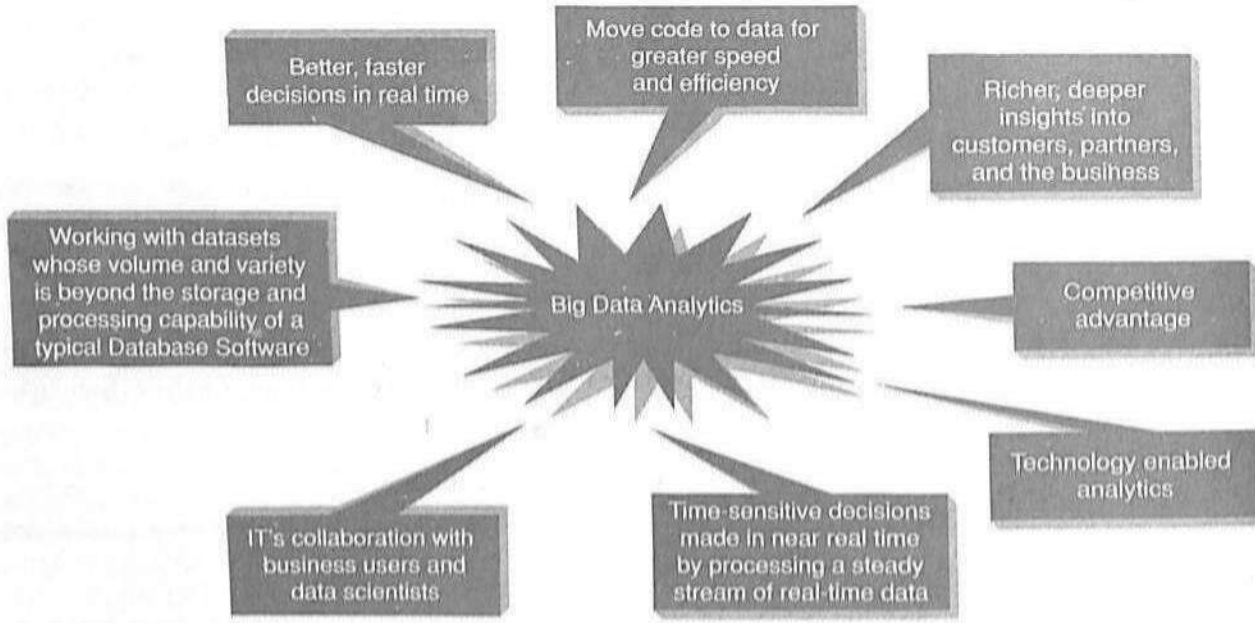


Figure 3.3 What is big data analytics?

Refer Figure 3.4. Big data isn't just about technology. It is about understanding what the data is saying to us. It is about understanding relationships that we thought never existed between datasets. It is about patterns and trends waiting to be unveiled.

And of course, big data analytics is not here to replace our now very robust and powerful Relational Database Management System (RDBMS) or our traditional Data Warehouse. It is here to coexist with both RDBMS and Data Warehouse, leveraging the power of each to yield business value. Big data analytics is not "One-size fits all" traditional RDBMS built on shared disk and memory.

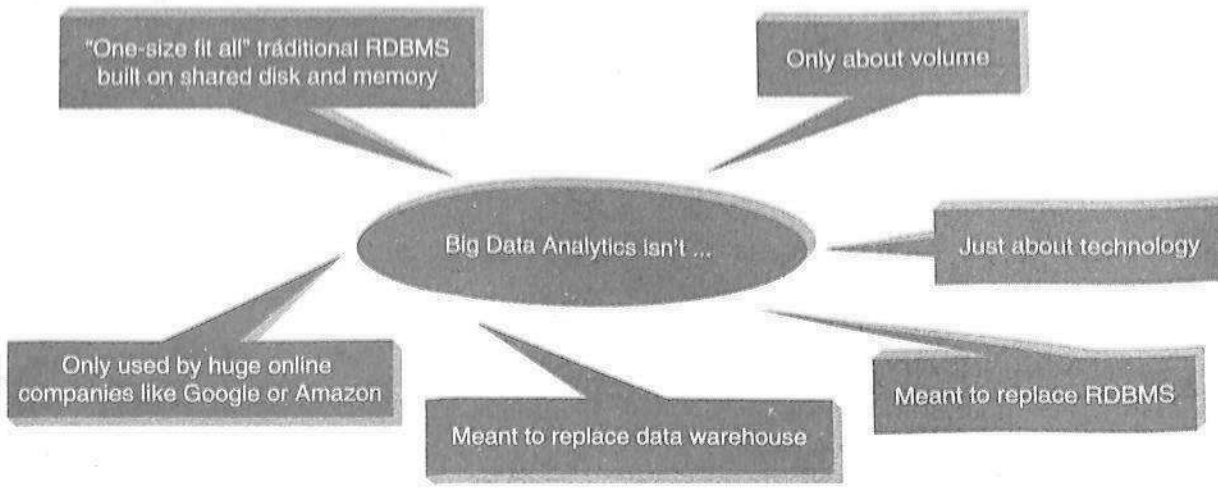


Figure 3.4 What big data analytics isn't?

And before we think it is only used by huge online companies like a Google or Amazon, let us clear the myth. It is for any business and any industry that needs actionable insights out of their data (both internal and external).

3.4 WHY THIS SUDDEN HYPE AROUND BIG DATA ANALYTICS?

If we go by the industry buzz, every place there seems to be talk about big data and big data analytics. Why this sudden hype? Refer Figure 3.5.

Let us put it down to three foremost reasons:

1. Data is growing at a 40% compound annual rate, reaching nearly 45 ZB by 2020. In 2010, almost about 1.2 trillion Gigabyte of data was generated. This amount doubled to 2.4 trillion Gigabyte in 2012 and to about 5 trillion Gigabytes in the year 2014. The volume of business data worldwide is expected to double every 1.2 years. Wal-Mart, the world retailer, processes one million customer transactions per hour. 500 million "tweets" are posted by Twitter users every day. 2.7 billion "Likes" and comments are posted by Facebook users in a day. Every day 2.5 quintillion bytes of data is created, with 90% of the world's data created in the past 2 years alone.
   Source:
   (a) http://www.intel.com/content/www/us/en/communications/internet-minute-infographic.html
   (b) http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html
2. Cost per gigabyte of storage has hugely dropped.
3. There are an overwhelming number of user-friendly analytics tools available in the market today.

3.5 CLASSIFICATION OF ANALYTICS

There are basically two schools of thought:

1. Those that classify analytics into basic, operationalized, advanced, and monetized.
2. Those that classify analytics into analytics 1.0, analytics 2.0, and analytics 3.0.

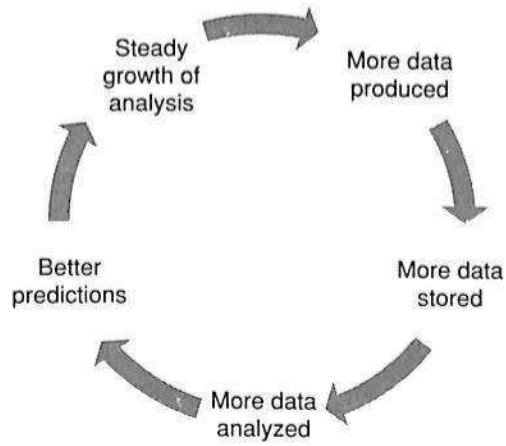


Figure 3.5 What big data entails?

3.5.1 First School of Thought

1. **Basic analytics:** This primarily is slicing and dicing of data to help with basic business insights. This is about reporting on historical data, basic visualization, etc.
2. **Operationalized analytics:** It is operationalized analytics if it gets woven into the enterprise's business processes.
3. **Advanced analytics:** This largely is about forecasting for the future by way of predictive and prescriptive modeling.
4. **Monetized analytics:** This is analytics in use to derive direct business revenue.

3.5.2 Second School of Thought

Let us take a closer look at analytics 1.0, analytics 2.0, and analytics 3.0. Refer Table 3.1.

Table 3.1 Analytics 1.0, 2.0, and 3.0

Analytics 1.0	Analytics 2.0	Analytics 3.0
Era: mid 1950s to 2009	**2005 to 2012**	**2012 to present**
Descriptive statistics (report on events, occurrences, etc. of the past)	Descriptive statistics + predictive statistics (use data from the past to make predictions for the future)	Descriptive + predictive + prescriptive statistics (use data from the past to make prophecies for the future and at the same time make recommendations to leverage the situation to one's advantage)
Key questions asked: What happened? Why did it happen?	Key questions asked: What will happen? Why will it happen?	Key questions asked: What will happen? When will it happen? Why will it happen? What should be the action taken to take advantage of what will happen?
Data from legacy systems, ERP, CRM, and 3rd party applications.	Big data	A blend of big data and data from legacy systems, ERP, CRM, and 3rd party applications.
Small and structured data sources. Data stored in enterprise data warehouses or data marts.	Big data is being taken up seriously. Data is mainly unstructured, arriving at a much higher pace. This fast flow of data entailed that the influx of big volume data had to be stored and processed rapidly, often on massive parallel servers running Hadoop.	A blend of big data and traditional analytics to yield insights and offerings with speed and impact.
Data was internally sourced.	Data was often externally sourced.	Data is both being internally and externally sourced.
Relational databases	Database appliances, Hadoop clusters, SQL to Hadoop environments, etc.	In memory analytics, in database processing, agile analytical methods, machine learning techniques, etc.

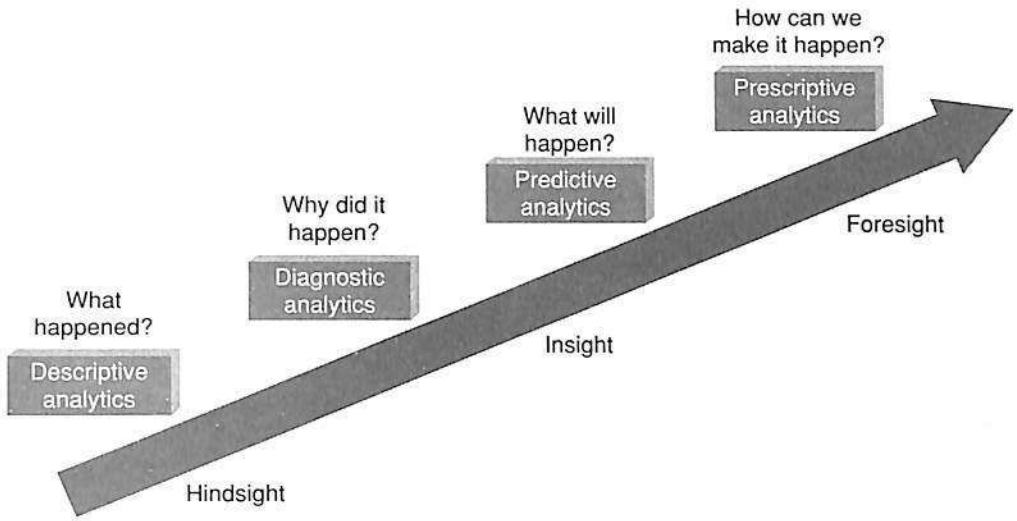


Figure 3.6 Analytics 1.0, 2.0, and 3.0.

Figure 3.6 shows the subtle growth of analytics from Descriptive → Diagnostic → Predictive → Prescriptive analytics.

3.6 GREATEST CHALLENGES THAT PREVENT BUSINESSES FROM CAPITALIZING ON BIG DATA

1. Obtaining executive sponsorships for investments in big data and its related activities (such as training, etc.).
2. Getting the business units to share information across organizational silos.
3. Finding the right skills (business analysts and data scientists) that can manage large amounts of structured, semi-structured, and unstructured data and create insights from it.
4. Determining the approach to scale rapidly and elastically. In other words, the need to address the storage and processing of large volume, velocity, and variety of big data.
5. Deciding whether to use structured or unstructured, internal or external data to make business decisions.
6. Choosing the optimal way to report findings and analysis of big data (visual presentation and analytics) for the presentations to make the most sense.
7. Determining what to do with the insights created from big data.

3.7 TOP CHALLENGES FACING BIG DATA

1. **Scale:** Storage (RDBMS (Relational Database Management System) or NoSQL (Not only SQL)) is one major concern that needs to be addressed to handle the need for scaling rapidly and elastically. The need of the hour is a storage that can best withstand the onslaught of large volume, velocity, and variety of big data? Should you scale vertically or should you scale horizontally?

2. **Security:** Most of the NoSQL big data platforms have poor security mechanisms (lack of proper authentication and authorization mechanisms) when it comes to safeguarding big data. A spot that cannot be ignored given that big data carries credit card information, personal information, and other sensitive data.
3. **Schema:** Rigid schemas have no place. We want the technology to be able to fit our big data and not the other way around. The need of the hour is dynamic schema. Static (pre-defined schemas) are passé.
4. **Continuous availability:** The big question here is how to provide 24/7 support because almost all RDBMS and NoSQL big data platforms have a certain amount of downtime built in.
5. **Consistency:** Should one opt for consistency or eventual consistency?
6. **Partition tolerant:** How to build partition tolerant systems that can take care of both hardware and software failures?
7. **Data quality:** How to maintain data quality – data accuracy, completeness, timeliness, etc.? Do we have appropriate metadata in place?

3.8 WHY IS BIG DATA ANALYTICS IMPORTANT?

Let us study the various approaches to analysis of data and what it leads to.

1. **Reactive – Business Intelligence:** What does Business Intelligence (BI) help us with? It allows the businesses to make faster and better decisions by providing the right information to the right person at the right time in the right format. It is about analysis of the past or historical data and then displaying the findings of the analysis or reports in the form of enterprise dashboards, alerts, notifications, etc. It has support for both pre-specified reports as well as ad hoc querying.
2. **Reactive – Big Data Analytics:** Here the analysis is done on huge datasets but the approach is still reactive as it is still based on static data.
3. **Proactive – Analytics:** This is to support futuristic decision making by the use of data mining, predictive modeling, text mining, and statistical analysis. This analysis is not on big data as it still uses the traditional database management practices on big data and therefore has severe limitations on the storage capacity and the processing capability.
4. **Proactive – Big Data Analytics:** This is sieving through terabytes, petabytes, exabytes of information to filter out the relevant data to analyze. This also includes high performance analytics to gain rapid insights from big data and the ability to solve complex problems using more data.

3.9 WHAT KIND OF TECHNOLOGIES ARE WE LOOKING TOWARD TO HELP MEET THE CHALLENGES POSED BY BIG DATA?

1. The first requirement is of cheap and abundant storage.
2. We need faster processors to help with quicker processing of big data.
3. Affordable open-source, distributed big data platforms, such as Hadoop.
4. Parallel processing, clustering, virtualization, large grid environments (to distribute processing to a number of machines), high connectivity, and high throughputs rather than low latency.
5. Cloud computing and other flexible resource allocation arrangements.

3.10 DATA SCIENCE

Data science is the science of extracting knowledge from data. In other words, it is a science of drawing out hidden patterns amongst data using statistical and mathematical techniques. It employs techniques and theories drawn from many fields from the broad areas of mathematics, statistics, information technology including machine learning, data engineering, probability models, statistical learning, pattern recognition and learning, etc.

Today we have a plethora of use-cases for "Data Science" that are already exploring massive datasets (Peta to Zetta bytes of Information) for weather predictions, oil drillings, seismic activities, financial frauds, terrorist network and activities, global economic impacts, sensor logs, social media analytics, and so many others beyond standard retail, manufacturing use-cases such as customer churn, market basket analytics (associative mining), collaborative filtering, regression analysis, etc. Data science is multi-disciplinary. Refer to Figure 3.7.

3.10.1 Business Acumen Skills

A data scientist should have the prowess to counter the pressures of business. A firm understanding of business domain further helps. The following is a list of traits that needs to be honed to play the role of data scientist.

1. Understanding of domain.
2. Business strategy.
3. Problem solving.
4. Communication.
5. Presentation.
6. Inquisitiveness.

3.10.2 Technology Expertise

It goes without saying that technology expertise will come in handy if one is to play the role of a data scientist. Cited below are few skills required as far as technical expertise is concerned.

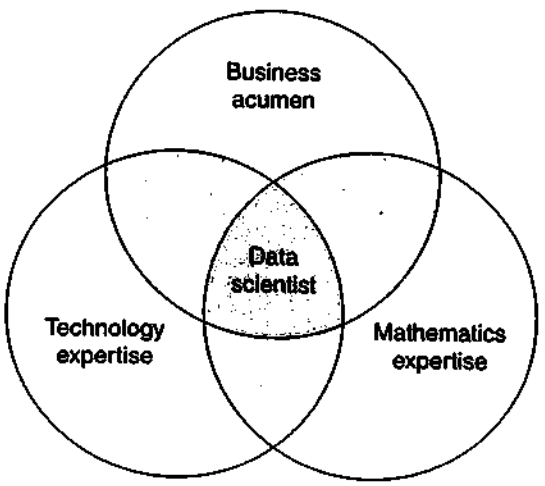


Figure 3.7 Data scientist.

1. Good database knowledge such as RDBMS.
2. Good NoSQL database knowledge such as MongoDB, Cassandra, HBase, etc.
3. Programming languages such as Java, Python, C++, etc.
4. Open-source tools such as Hadoop.
5. Data warehousing.
6. Data mining.
7. Visualization such as Tableau, Flare, Google visualization APIs, etc.

3.10.3 Mathematics Expertise

Since the core job of the data scientist will require him to comprehend data, interpret it, make sense of it, and analyze it, he/she will have to dabble in learning algorithms. The following are the key skills that a data scientist will have to have in his arsenal.

1. Mathematics.
2. Statistics.
3. Artificial Intelligence (AI).
4. Algorithms.
5. Machine learning.
6. Pattern recognition.
7. Natural Language Processing.

To sum it up, the data science process is

1. Collecting raw data from multiple disparate data sources.
2. Processing the data.
3. Integrating the data and preparing clean datasets.
4. Engaging in explorative data analysis using model and algorithms.
5. Preparing presentations using data visualizations (commonly called Infographics, or BizAnalytics, or VizAnalytics, etc.)
6. Communicating the findings to all stakeholders.
7. Making faster and better decisions.

3.11 DATA SCIENTIST...YOUR NEW BEST FRIEND!!!

In today's data age, a data scientist is the best friend that you can gift yourself. Refer Figure 3.8 to learn about the tasks that the data scientist can help you with.

3.11.1 Responsibilities of a Data Scientist

Refer Figure 3.8.

1. **Data Management:** A data scientist employs several approaches to develop the relevant datasets for analysis. Raw data is just "RAW," unsuitable for analysis. The data scientist works on it to prepare it to reflect the relationships and contexts. This data then becomes useful for processing and further analysis.

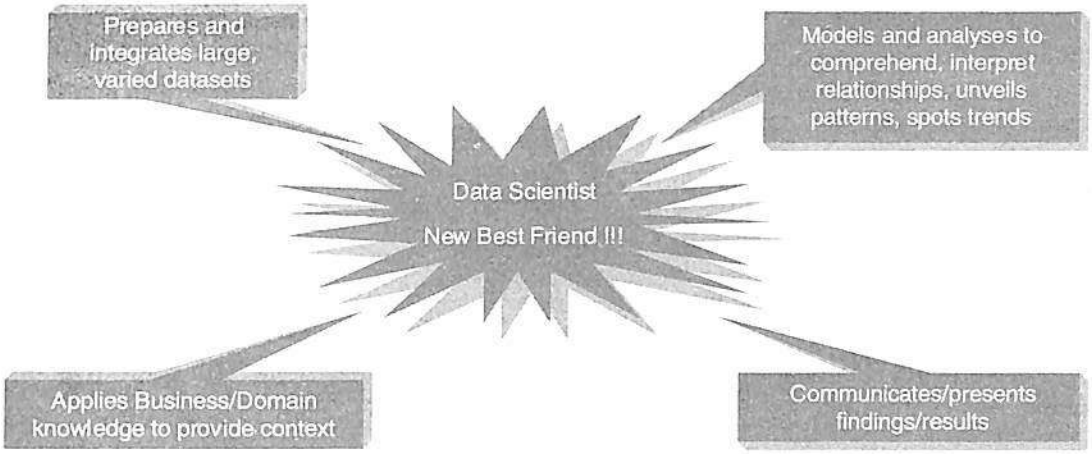


Figure 3.8 Data scientist: your new best friend!!!

2. **Analytical Techniques:** Depending on the business questions which we are trying to find answers to and the type of data available at hand, the data scientist employs a blend of analytical techniques to develop models and algorithms to understand the data, interpret relationships, spot trends, and unveil patterns.
3. **Business Analysts:** A data scientist is a business analyst who distinguishes cool facts from insights and is able to apply his business acumen and domain knowledge to see the results in the business context. He is a good presenter and communicator who is able to communicate the results of his findings in a language that is understood by the different business stakeholders.

3.12 TERMINOLOGIES USED IN BIG DATA ENVIRONMENTS

In order to get a good handle on the big data environment, let us get familiar with a few key terminologies in this arena.

3.12.1 In-Memory Analytics

Data access from non-volatile storage such as hard disk is a slow process. The more the data is required to be fetched from hard disk or secondary storage, the slower the process gets. One way to combat this challenge is to pre-process and store data (cubes, aggregate tables, query sets, etc.) so that the CPU has to fetch a small subset of records. But this requires thinking in advance as to what data will be required for analysis. If there is a need for different or more data, it is back to the initial process of pre-computing and storing data or fetching it from secondary storage.

This problem has been addressed using in-memory analytics. Here all the relevant data is stored in Random Access Memory (RAM) or primary storage thus eliminating the need to access the data from hard disk. The advantage is faster access, rapid deployment, better insights, and minimal IT involvement.

3.12.2 In-Database Processing

In-database processing is also called as *in-database analytics*. It works by fusing data warehouses with analytical systems. Typically the data from various enterprise On Line Transaction Processing (OLTP) systems after

cleaning up (de-duplication, scrubbing, etc.) through the process of ETL is stored in the Enterprise Data Warehouse (EDW) or data marts. The huge datasets are then exported to analytical programs for complex and extensive computations. With in-database processing, the database program itself can run the computations eliminating the need for export and thereby saving on time. Leading database vendors are offering this feature to large businesses.

3.12.3 Symmetric Multiprocessor System (SMP)

In SMP, there is a single common main memory that is shared by two or more identical processors. The processors have full access to all I/O devices and are controlled by a single operating system instance.

SMP are tightly coupled multiprocessor systems. Each processor has its own high-speed memory, called cache memory and are connected using a system bus. Refer Figure 3.9.

3.12.4 Massively Parallel Processing

Massive Parallel Processing (MPP) refers to the coordinated processing of programs by a number of processors working parallel. The processors, each have their own operating systems and dedicated memory. They work on different parts of the same program. The MPP processors communicate using some sort of messaging interface. The MPP systems are more difficult to program as the application must be divided in such a way that all the executing segments can communicate with each other. MPP is different from Symmetrically Multiprocessing (SMP) in that SMP works with the processors sharing the same operating system and same memory. SMP is also referred to as *tightly-coupled multiprocessing.*

3.12.5 Difference Between Parallel and Distributed Systems

The next two terms that we discuss are parallel and distributed systems.

As is evident from Figure 3.10, a parallel database system is a tightly coupled system. The processors co-operate for query processing. The user is unaware of the parallelism since he/she has no access to a specific

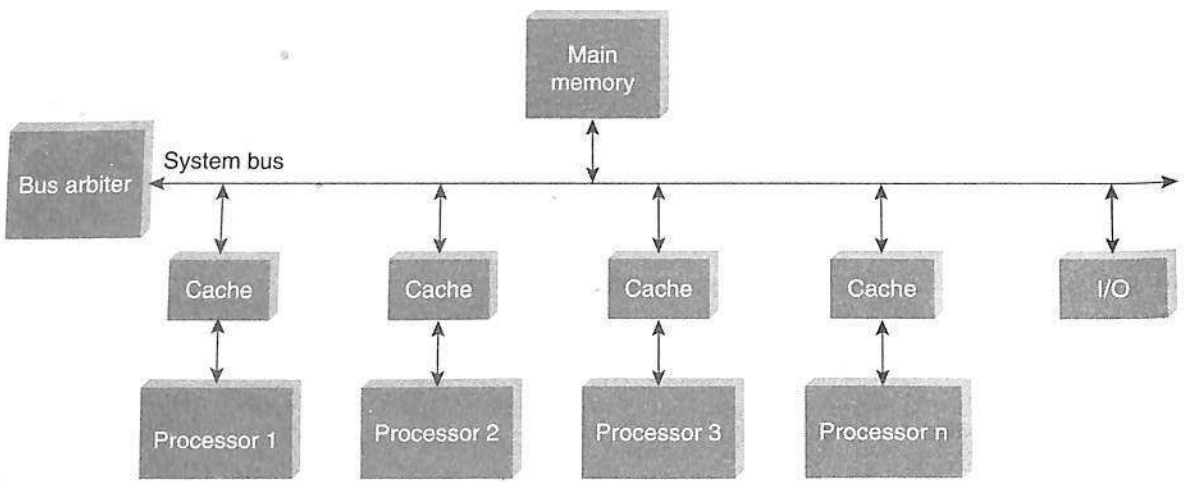


Figure 3.9 Symmetric Multiprocessor System.

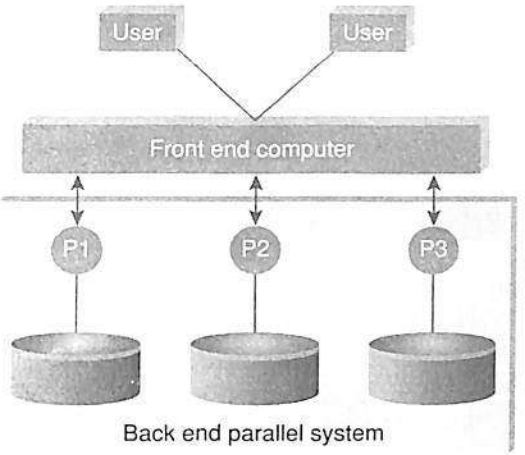


Figure 3.10 Parallel system.

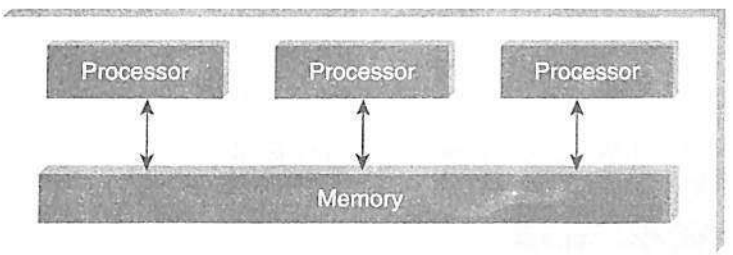


Figure 3.11 Parallel system.

processor of the system. Either the processors have access to a common memory (Refer Fig 3.11) or make use of message passing for communication.

Distributed database systems are known to be loosely coupled and are composed by individual machines. Refer Figure 3.12. Each of the machines can run their individual application and serve their own respective user. The data is usually distributed across several machines, thereby necessitating quite a number of machines to be accessed to answer a user query. Refer Figure 3.13.

3.12.6 Shared Nothing Architecture

Let us look at the three most common types of architecture for multiprocessor high transaction rate systems. They are:

1. Shared Memory (SM).
2. Shared Disk (SD).
3. Shared Nothing (SN).

In shared memory architecture, a common central memory is shared by multiple processors. In shared disk architecture, multiple processors share a common collection of disks while having their own private memory. In shared nothing architecture, neither memory nor disk is shared among multiple processors.

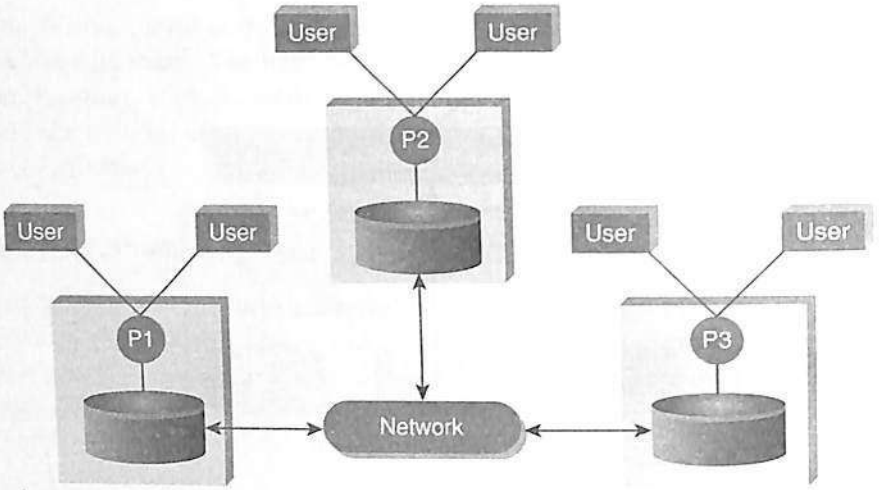


Figure 3.12 Distributed system.

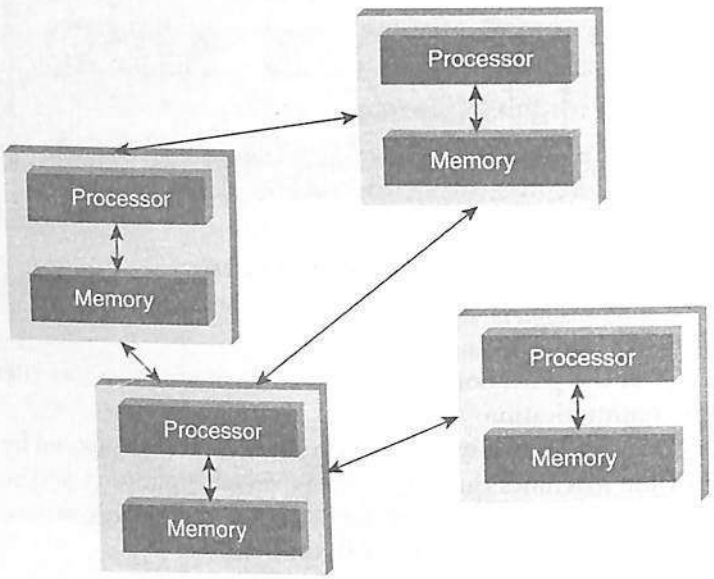


Figure 3.13 Distributed system.

3.12.6.1 Advantages of a "Shared Nothing Architecture"

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

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

3.12.7 CAP Theorem Explained

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

1. Consistency
2. Availability
3. Partition tolerance

3.12.7.1 CAP Theorem

Let us spend some time understanding the earlier mentioned terms.

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

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

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

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

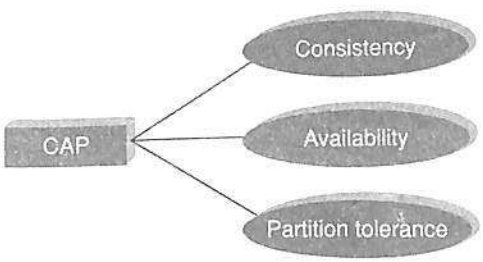


Figure 3.14 Brewer's CAP.

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

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

You: Hey Amey!

Amey: Hi! How can I help?

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

Amey: Sure! Just a minute.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

When to choose consistency over availability and vice-versa...

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

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

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

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

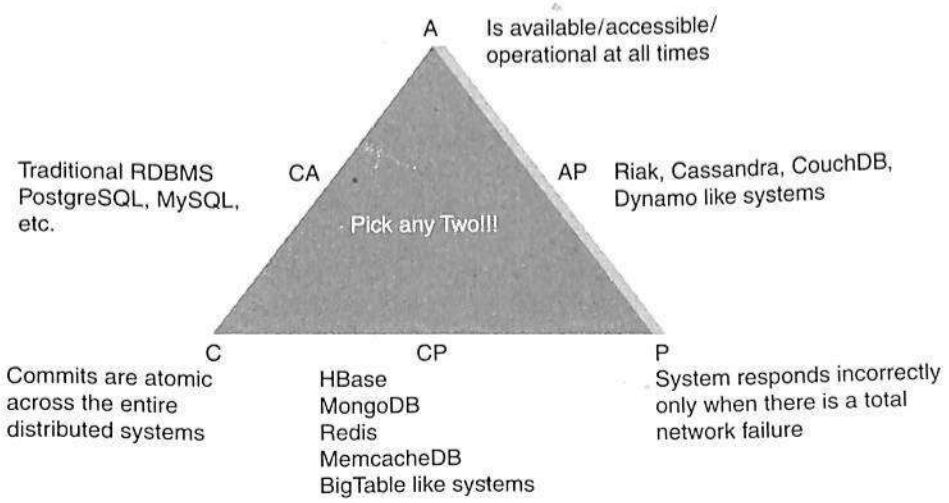


Figure 3.15 Databases and CAP.

3.13 BASICALLY AVAILABLE SOFT STATE EVENTUAL CONSISTENCY (BASE)

A few basic questions to start with:

1. ***Where is it used?***
   In distributed computing.
2. ***Why is it used?***
   To achieve high availability.
3. ***How is it achieved?***
   Assume a given data item. If no new updates are made to this given data item for a stipulated period of time, eventually all accesses to this data item will return the updated value. In other words, if no new updates are made to a given data item for a stipulated period of time, all updates that were made in the past and not applied to this given data item and the several replicas of it will percolate to this data item so that it stays as current/recent as is possible.
4. ***What is replica convergence?***
   A system that has achieved eventual consistency is said to have converged or achieved *replica convergence.*
5. ***Conflict resolution*: How is the conflict resolved?**
   (a) **Read repair:** If the read leads to discrepancy or inconsistency, a correction is initiated. It slows down the read operation.
   (b) **Write repair:** If the write leads to discrepancy or inconsistency, a correction is initiated. This will cause the write operation to slow down.
   (c) **Asynchronous repair:** Here, the correction is not part of a read or write operation.

3.14 FEW TOP ANALYTICS TOOLS

There is no dearth of analytical tools in the market. Please find below our list of few top analytics tools. We have also provided the links after each tool for you to explore more...

1. MS Excel
   https://support.office.microsoft.com/en-in/article/Whats-new-in-Excel-2013-1cbc42cd-bfaf-43d7-9031-5688ef1392fd?CorrelationId=1a2171cc-191f-47de-8a55-08a5f2e9c739&ui=en-US&rs=en-IN&ad=IN
2. SAS
   http://www.sas.com/en_us/home.html
3. IBM SPSS Modeler
   http://www-01.ibm.com/software/analytics/spss/products/modeler/
4. Statistica
   http://www.statsoft.com/

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

3.14.1 Open Source Analytics Tools

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

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

REMIND ME

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

CONNECT ME (INTERNET RESOURCES)

- http://en.wikipedia.org/wiki/Data_science
- http://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/
- http://www.oralytics.com/2012/06/data-science-is-multidisciplinary.html
- http://spotfire.tibco.com/blog/?p=4240
- http://reports.informationweek.com/abstract/106/1255/Financial/tech-center-taking-advantage-of-in-memory-analytics.html
- http://www.informationweek.com/software/information-management/oracle-analytics-package-expands-in-database-processing-options/d/d-id/1102712?

TEST ME

A. Fill Me

1. The _____ technology helps query data that resides in a computer's random access memory (RAM) rather than data stored on physical disks.
2. Eventual consistency is a consistency model used in distributed computing to achieve high _____.
3. A coordinated processing of a program by multiple processors, each working on different parts of the program and using its own operating system and memory is called _____.
4. A collection of independent computers that appear to its users as a single coherent system is _____.

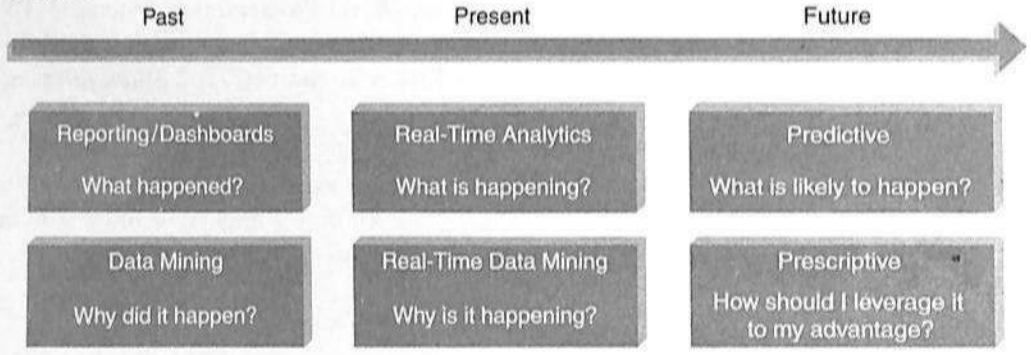
Answers:

1. In-memory analytics
2. Availability
3. Massively parallel processing
4. Distributed systems

B. Answer Me

1. What are the various types of analytics?

Answer:



2. What are the key questions to be answered by all organizations stepping into analytics?

Answer: The key questions for any organization stepping into analytics are:

- Should you be storing all of your big data? If "Yes", where are you going to store it? If "No", how will you know what to store and what to discard?
- How will you sieve through your massive data to filter out the relevant from the irrelevant?
- How long will you store this data?
- How will you accommodate the peaks (variability in terms of data influx) in your data?
- How will you analyze? Will you analyze all the data that is stored or analyze a sample?
- What will you do with the insights generated from this analysis?

3. What can one expect from analytics 3.0?

Answer:

- In-memory analytics.
- In-database processing.
- Leveraging analytics to improve operational, tactical, and strategic decision making.

- Coupling the in-memory analytics and in-database processing with agile analytical methods and machine learning techniques.
- Appropriate tools to effectively support decision-making at the front lines, such as mobile and self-serve analytical applications.

4. Which industries will be affected most by analytics 3.0? Who will benefit the most?

   Answer: Almost all the firms in all the industries and not just online firms will be affected by analytics 3.0. A lot of analytics have already been done in the Transport, Retail, and Banking sector. Telecom, entertainment, and health sectors have a bit of catching up to do.

5. What is predictive and prescriptive analytics?

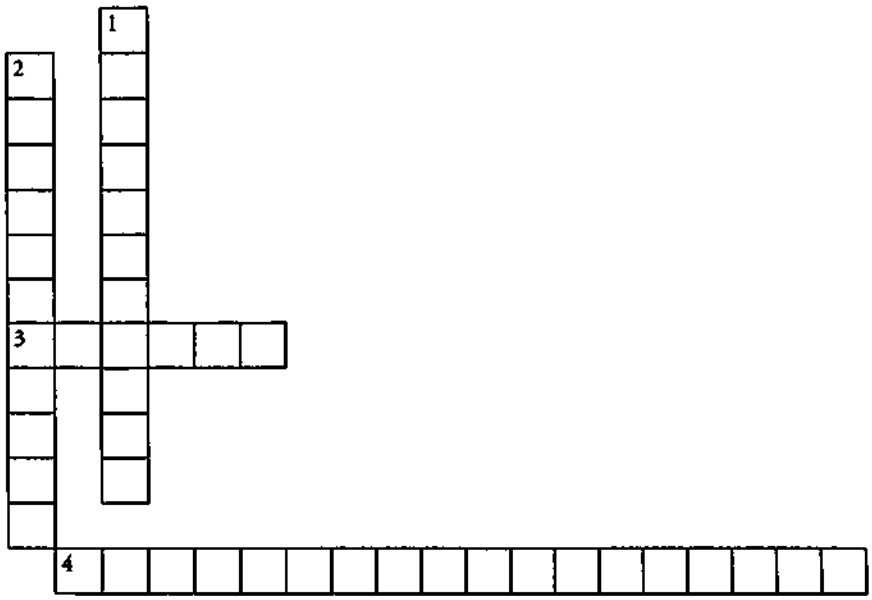
   Answer:

   Predictive analytics helps you answer the questions: "What will happen?" and "Why will it happen"?

   Prescriptive analytics goes beyond "What will happen?" "Why will it happen?" and "When will it happen?" to answer "What should be the action taken to take advantage of what will happen"?

C. Crossword

1. Puzzle on CAP Theorem



Across

3. CAP theorem is also called as _____ theorem.
4. System will continue to function even when network partition occurs.

Down

1. Every read fetches the most recent write.
2. A non-failing node will return a reasonable response within a reasonable amount of time.

Solution:

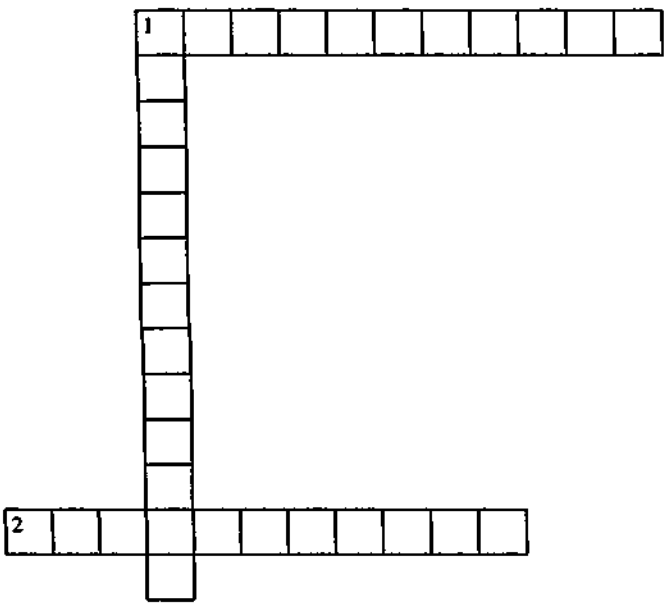
Across

3. Brewer
4. Partition Tolerant

Down

1. Consistency
2. Availability

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, relational database that did not expose the standard SQL interface. Johan Oskarsson, who was then a developer at last.fm, in 2009 reintroduced the term NoSQL at an event called to discuss open-source distributed network. The #NoSQL was coined by Eric Evans and few other database people at the event found it suitable to describe these non-relational databases.

Few features of NoSQL databases are as follows:

1. They are open source.
2. They are non-relational.
3. They are distributed.
4. They are schema-less.
5. They are cluster friendly.
6. They are born out of 21st century web applications.

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. NoSQL databases.

1. **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.

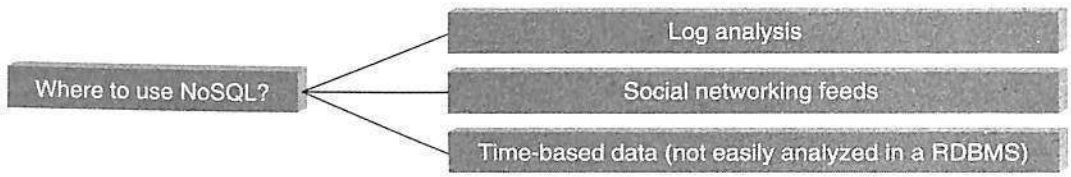


Figure 4.1 Where to use NoSQL?

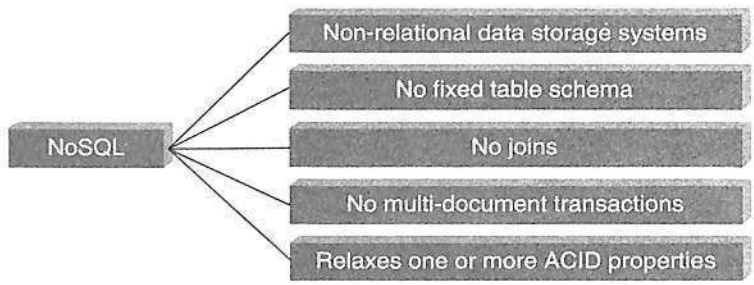


Figure 4.2 What is NoSQL?

2. **Are distributed:** They are distributed meaning the data is distributed across several nodes in a cluster constituted of low-cost commodity hardware.
3. **Offer 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. **Provide 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.

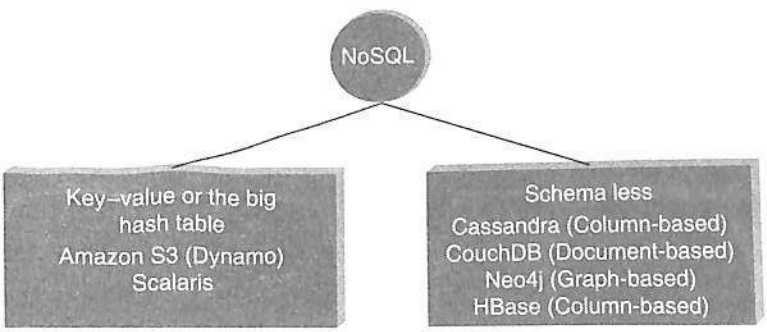
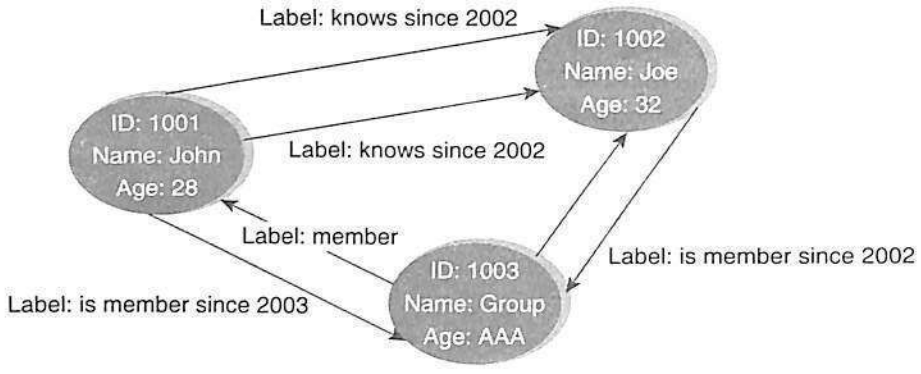


Figure 4.3 Types of NoSQL databases.

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

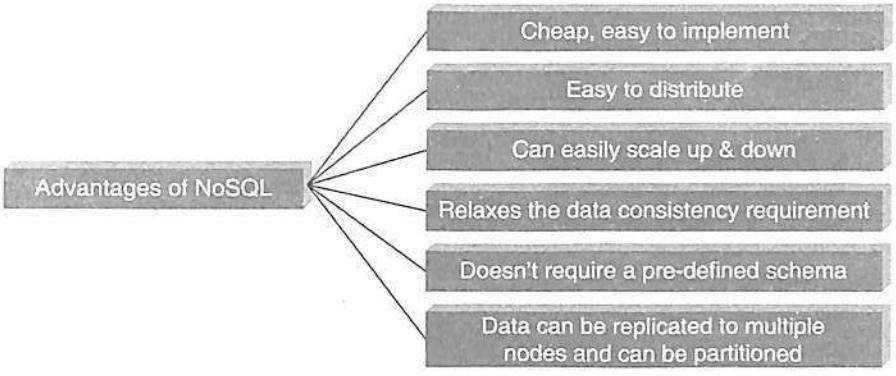


Figure 4.4 Advantages of NoSQL.

(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; this means that 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.

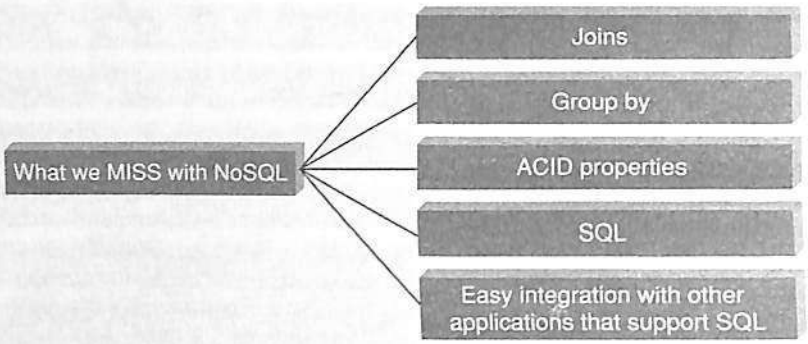


Figure 4.5 What we miss with NoSQL?

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.

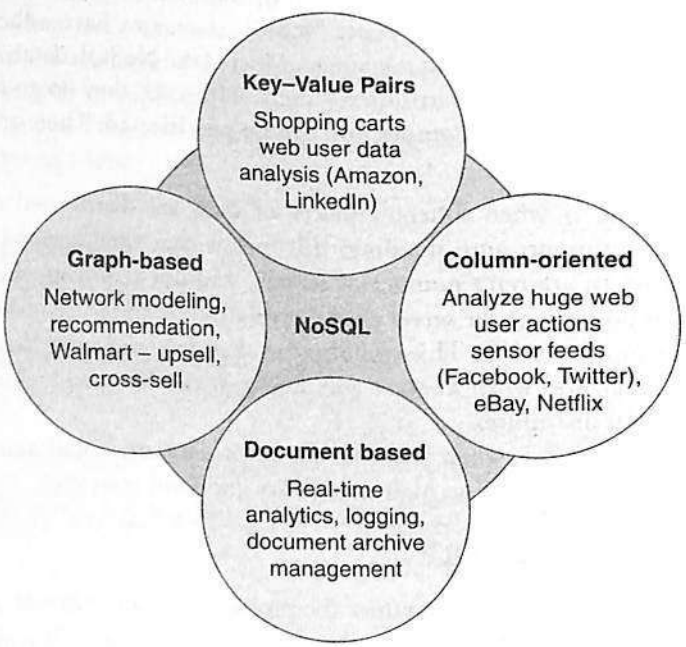


Figure 4.6 Use of NoSQL in industry.

4.1.8 NoSQL Vendors

Refer Table 4.2 for few popular NoSQL vendors.

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

4.1.9 SQL versus NoSQL

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

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), some others can be configured for eventual consistency (e.g., Cassandra)
Examples: Oracle, DB2, MySQL, MS SQL, PostgreSQL, etc.	Examples: MongoDB, HBase, Cassandra, Redis, Neo4j, CouchDB, Couchbase, Riak, etc.

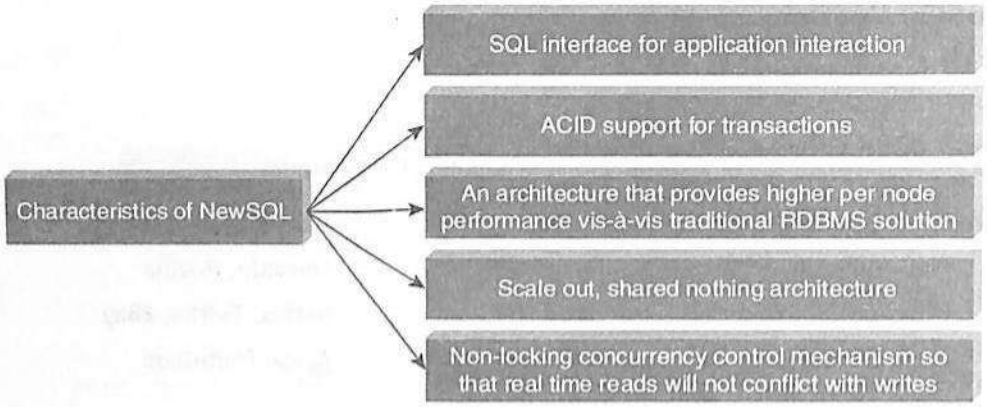


Figure 4.7 Characteristics of NewSQL.

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

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, Twitter, etc. Refer Figure 4.8.

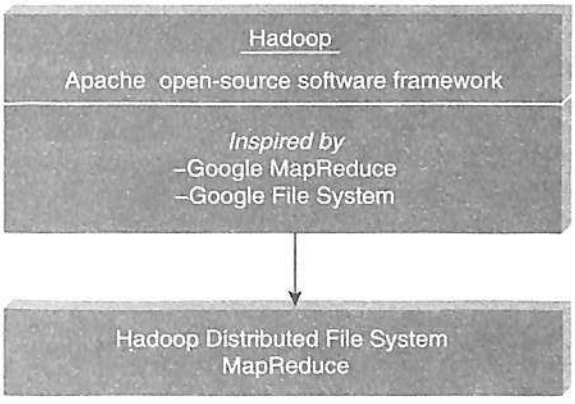


Figure 4.8 Hadoop.

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 datasets.

4.2.2 Key Advantages of Hadoop

Refer Figure 4.9 for a quick look at the key advantages of Hadoop. Some of them are as follows:

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.
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.

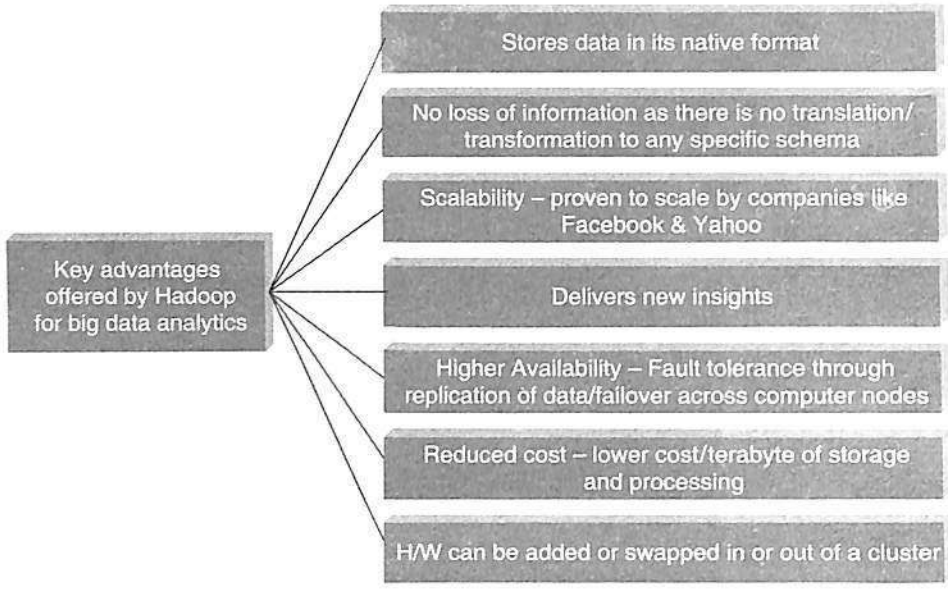


Figure 4.9 Key advantages of Hadoop.

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:** Processing is extremely fast in Hadoop as compared to other conventional systems owing to the "move code to data" paradigm.

   Hadoop has a shared-nothing architecture.

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. These data files can be in just about any

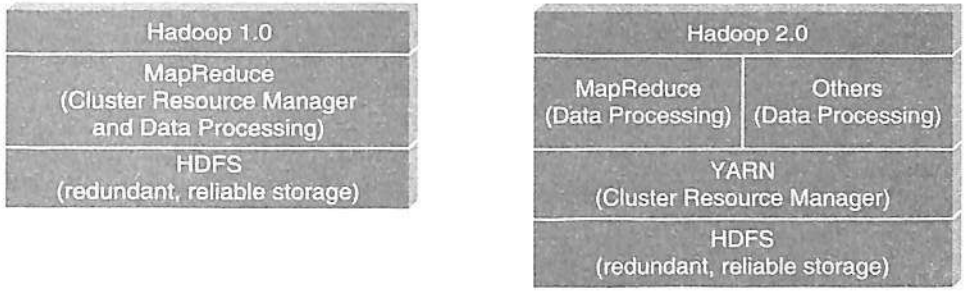


Figure 4.10 Versions of Hadoop.

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.

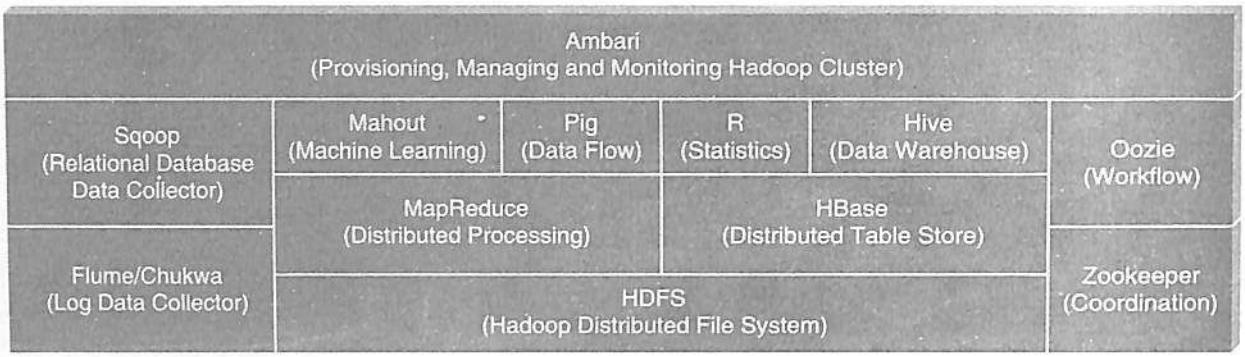


Figure 4.11 Hadoop ecosystem.

4.2.4 Overview of Hadoop Ecosystems

The components of the Hadoop ecosystem are shown in Figure 4.11.

There are components available in the Hadoop ecosystem for data ingestion, processing, and analysis.

Data Ingestion → Data Processing → Data Analysis

Components that help with Data Ingestion are:

1. Sqoop
2. Flume

Components that help with Data Processing are:

1. MapReduce
2. Spark

Components that help with Data Analysis are:

1. Pig
2. Hive
3. Impala

HDFS

It is the distributed storage unit of Hadoop. It provides streaming access to file system data as well as file permissions and authentication. It is based on GFS (Google File System). It is used to scale a single cluster node to hundreds and thousands of nodes. It handles large datasets running on commodity hardware. HDFS is highly fault-tolerant. It stores files across multiple machines. These files are stored in redundant fashion to allow for data recovery in case of failure.

PICTURE THIS...

An e-commerce website stores millions of customers' data in a distributed manner. Data has been collected over 4–5 years. It then runs batch analytics on the archived data to analyze customer's behavior, buying patterns, their preferences, their requirements, etc. This helps to understand which products are purchased by customers in which months, etc.

HBase

It stores data in HDFS. It is the first non-batch component of the Hadoop Ecosystem. It is a database on top of HDFS. It provides a quick random access to the stored data. It has very low latency compared to HDFS. It is a NoSQL database, is non-relational and is a column-oriented database. A table can have thousands of columns. A table can have multiple rows. Each row can have several column families. Each column family can have several columns. Each column can have several key values. It is based on Google BigTable. This is widely used by Facebook, Twitter, Yahoo, etc.

PICTURE THIS...

The same e-commerce website as in the HDFS case above also stores millions of product data. To search for a product among millions of products and to produce the result immediately (or you can say in real time), it needs to optimize the request and search process. HBase supports real-time analytics.

Given the huge velocity of data, they opted for HBase over HDFS, as HDFS does not support real-time writes. The results were overwhelming; it reduced the query time from 3 days to 3 minutes.

Difference between HBase and Hadoop/HDFS

1. HDFS is the file system whereas HBase is a Hadoop database. It is like NTFS and MySQL.
2. HDFS is WORM (Write once and read multiple times or many times). Latest versions support appending of data but this feature is rarely used. However, HBase supports real-time random read and write.
3. HDFS is based on Google File System (GFS) whereas HBase is based on Google Big Table.
4. HDFS supports only full table scan or partition table scan. Hbase supports random small range scan or table scan.
5. Performance of Hive on HDFS is relatively very good but for HBase it becomes 4–5 times slower.
6. The access to data is via MapReduce job only in HDFS whereas in HBase the access is via Java APIs, Rest, Avro, Thrift APIs.
7. HDFS does not support dynamic storage owing to its rigid structure whereas HBase supports dynamic storage.
8. HDFS has high latency operations whereas HBase has low latency operations.
9. HDFS is most suitable for batch analytics whereas HBase is for real-time analytics.

Hadoop Ecosystem Components for Data Ingestion

1. **Sqoop:** Sqoop stands for SQL to Hadoop. Its main functions are
   a) Importing data from RDBMS such as MySQL, Oracle, DB2, etc. to Hadoop file system (HDFS, HBase, Hive).
   b) Exporting data from Hadoop File system (HDFS, HBase, Hive) to RDBMS (MySQL, Oracle, DB2).

   Uses of Sqoop

   a) It has a connector-based architecture to allow plug-ins to connect to external systems such as MySQL, Oracle, DB2, etc.

b) It can provision the data from external system on to HDFS and populate tables in Hive and HBase.
c) It integrates with Oozie allowing you to schedule and automate import and export tasks.

2. **Flume:** Flume is an important log aggregator (aggregates logs from different machines and places them in HDFS) component in the Hadoop ecosystem. Flume has been developed by Cloudera. It is designed for high volume ingestion of event-based data into Hadoop. The default destination in Flume (called as sink in flume parlance) is HDFS. However it can also write to HBase or Solr.

PICTURE THIS...

There is a bank of web servers. Flume moves log events from those files into new aggregated files in HDFS for processing.

Hadoop Ecosystem Components for Data Processing

1. **MapReduce:** It is a programing paradigm that allows distributed and parallel processing of huge datasets. It is based on Google MapReduce. Google released a paper on MapReduce programming paradigm in 2004 and that became the genesis of Hadoop processing model. The MapReduce framework gets the input data from HDFS. There are two main phases: Map phase and the Reduce phase. The map phase converts the input data into another set of data (key–value pairs). This new intermediate dataset then serves as the input to the reduce phase. The reduce phase acts on the datasets to combine (aggregate and consolidate) and reduce them to a smaller set of tuples. The result is then stored back in HDFS.
2. **Spark:** It is both a programming model as well as a computing model. It is an open-source big data processing framework. It was originally developed in 2009 at UC Berkeley's AmpLab and became an open-source project in 2010. It is written in Scala. It provides in-memory computing for Hadoop. In Spark, workloads execute in memory rather than on disk owing to which it is much faster (10 to 100 times) than when the workload is executed on disk. However, if the datasets are too large to fit into the available system memory, it can perform conventional disk-based processing. It serves as a potentially faster and more flexible alternative to MapReduce. It accesses data from HDFS (Spark does not have its own distributed file system) but bypasses the MapReduce processing.

   Spark can be used with Hadoop coexisting smoothly with MapReduce (sitting on top of Hadoop YARN) or used independently of Hadoop (standalone). As a programming model, it works well with Scala, Python (it has API connectors for using it with Java or Python) or R programming language.

   The following are the Spark libraries:

   a) ***Spark SQL:*** Spark also has support for SQL. Spark SQL uses SQL to help query data stored in disparate applications.
   b) ***Spark streaming:*** It helps to analyze and present data in real time.
   c) ***MLib:*** It supports machine learning such as applying advanced statistical operations on data in Spark Cluster.
   d) ***GraphX:*** It helps in graph parallel computation.

   Spark and Hadoop are usually used together by several companies. Hadoop was primarily designed to house unstructured data and run batch processing operations on it. Spark is used extensively for its

high speed in memory computing and ability to run advanced real-time analytics. The two together have been giving very good results.

Hadoop Ecosystem Components for Data Analysis

1. **Pig:** It is a high-level scripting language used with Hadoop. It serves as an alternative to MapReduce. It has two parts:
   *(a) **Pig Latin:*** It is SQL-like scripting language. Pig Latin scripts are translated into MapReduce jobs which can then run on YARN and process data in the HDFS cluster. It was initially developed by Yahoo. It is immensely popular with developers who are not comfortable with MapReduce. However, SQL developers may have a preference for Hive.

   How it works? There is a "Load" command available to load the data from "HDFS" into Pig. Then one can perform functions such as grouping, filtering, sorting, joining etc. The processed or computed data can then be either displayed on screen or placed back into HDFS.

   It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analyzing huge data sets.

   *(b) **Pig runtime:*** It is the runtime environment.
2. **Hive:** Hive is a data warehouse software project built on top of Hadoop. Three main tasks performed by Hive are summarization, querying and analysis. It supports queries written in a language called HQL or HiveQL which is a declarative SQL-like language. It converts the SQL-style queries into MapReduce jobs which are then executed on the Hadoop platform.

Difference between Hive and RDBMS

Both Hive and traditional databases such as MySQL, MS SQL Server, PostgreSQL support SQL interface. However, Hive is better known as a datawarehouse (D/W) rather than a database.

Let us look at the difference between Hive and traditional databases as regards the schema.

1. Hive enforces schema on Read Time whereas RDBMS enforces schema on Write Time. In RDBMS, at the time of loading/inserting data, the table's schema is enforced. If the data being loaded does not conform to the schema then it is rejected. Thus, the schema is enforced on write (loading the data into the database). Schema on write takes longer to load the data into the database; however it makes up for it during data retrieval with a good query time performance. However, Hive does not enforce the schema when the data is being loaded into the D/W. It is enforced only when the data is being read/retrieved. This is called schema on read. It definitely makes for fast initial load as the data load or insertion operation is just a file copy or move.
2. Hive is based on the notion of write once and read many times whereas the RDBMS is designed for read and write many times.
3. Hadoop is a batch-oriented system. Hive, therefore, is not suitable for OLTP (Online Transaction Processing) but, although not ideal, seems closer to OLAP (Online Analytical Processing). The reason being that there is quite a latency between issuing a query and receiving a reply as the query written in HiveQL will be converted to MapReduce jobs which are then executed on the Hadoop cluster. RDBMS is suitable for housing day-to-day transaction data and supports all OLTP operations with frequent insertions, modifications (updates), deletions of the data.

4. Hive handles static data analysis which is non-real-time data. Hive is the data warehouse of Hadoop. There are no frequent updates to the data and the query response time is not fast. RDBMS is suited for handling dynamic data which is real time.
5. Hive can be easily scaled at a very low cost when compared to RDMS. Hive uses HDFS to store data, thus it cannot be considered as the owner of the data, while on the other hand RDBMS is the owner of the data responsible for storing, managing and manipulating it in the database.
6. Hive uses the concept of parallel computing, whereas RDBMS uses serial computing.

We summarize the difference in Table 4.5.

Table 4.5 Hive versus RDBMS

	Hadoop	RDBMS
Data Variety	Used for structured, semi-structured and unstructured data. Hadoop supports a variety of data formats in real time such as XML, JSON, and text-based flat file formats.	Used for structured data
Data Storage	Usually datasets of size terabytes, petabytes	Usually datasets of size gigabytes
Querying	HiveQL	SQL
Query Response	In Hadoop, there is latency due to batch processing.	In RDBMS, query response time is immediate.
Schema	Schema required on read	Schema required on write
Speed	Writes are faster compared to reads as there is no adherence to schema required at the time of inserting or writing data. Schema is enforced at read time	Reads are very fast (supported by building indexes on required columns).
	Hadoop is designed for write once read many times. It does not work for random reading and writing of a few records like RDBMS.	RDBMS is designed for read and write many times.
Cost	Apache Hadoop is open-source, large-scale, distributed, scalable, data intensive computing.	Available as proprietary RDBMS such as Oracle, MS SQL Server, IBM DB2, etc. Also open-source RDBMS are available such as MySQL, PostgreSQL, etc.
Use Cases	Analytics, data discovery	OLTP (Online Transaction Processing). Mainly used to store and process day-to-day business data.

(Continued)

Table 4.5 (Continued)

	Hadoop	RDBMS
Throughput	High	Low
Scalability	Horizontal (Hadoop scales by adding nodes to a Hadoop cluster of easily available commodity machines).	Vertical: RDBMS scales vertically by increasing the horsepower (CPU, Hard Disk Capacity, RAM, etc.) of the machine.
Hardware	Commodity/Utility Hardware	High End Servers
Integrity	Low	High. Obeys ACID properties A – Atomicity C – Consistency I – Integrity D – Durability

Difference between Hive and HBase

1. Hive is a MapReduce-based SQL engine that runs on top of Hadoop. HBase is a key–value NoSQL database that runs on top of HDFS.
2. Hive is for batch processing of big data. HBase is for real-time data streaming.

Impala

It is a high performance SQL engine that runs on Hadoop cluster. It is ideal for interactive analysis. It has very low latency measured in milliseconds. It supports a dialect of SQL called Impala SQL.

ZooKeeper

It is a coordination service for distributed applications.

Oozie

It is a workflow scheduler system to manage Apache Hadoop jobs.

Mahout

It is a scalable machine learning and data mining library.

Chukwa

It is a data collection system for managing large distributed systems.

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.

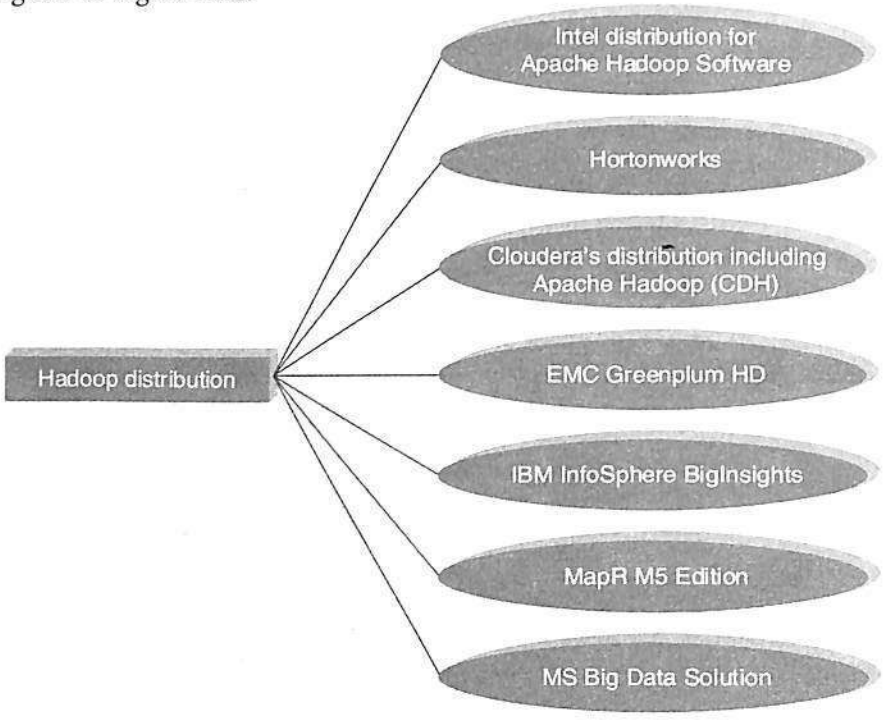


Figure 4.12 Hadoop distributions.

4.2.6 Hadoop versus SQL

Table 4.6 lists the differences between Hadoop and SQL.

Table 4.6 Hadoop versus SQL

Hadoop	SQL
Scale out	Scale up
Key–Value pairs	Relational table
Functional Programming	Declarative Queries
Offline batch processing	Online 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.

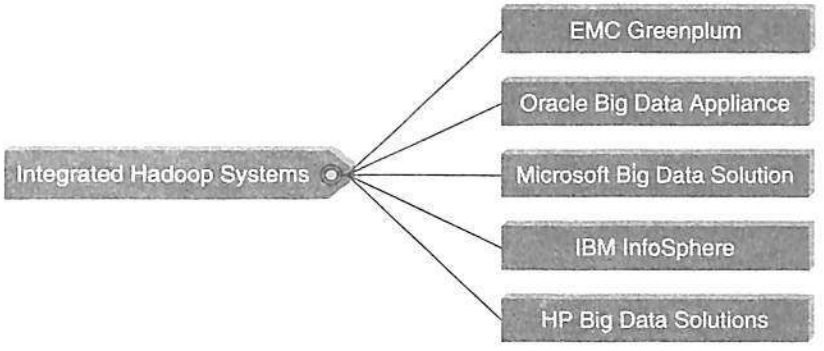


Figure 4.13 Integrated Hadoop systems.

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.

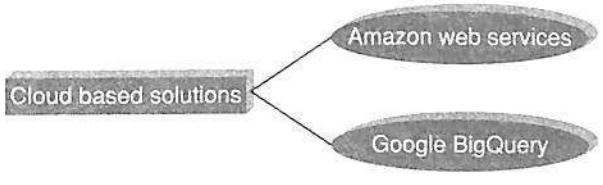


Figure 4.14 Cloud-based solutions.

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 Yet Another Resource Negotiator (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

"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.

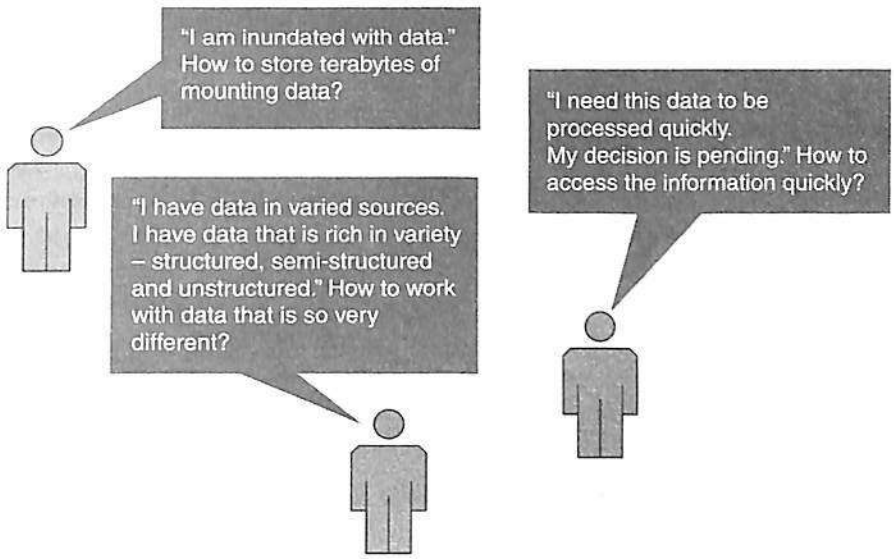


Figure 5.1 Challenges with big volume, variety, and velocity of data.

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.

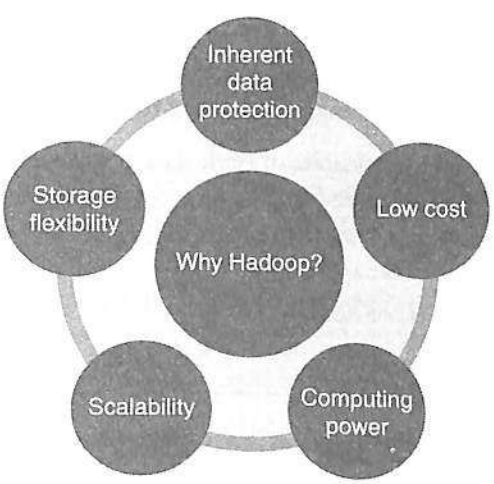


Figure 5.2 Key considerations of Hadoop.

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**.

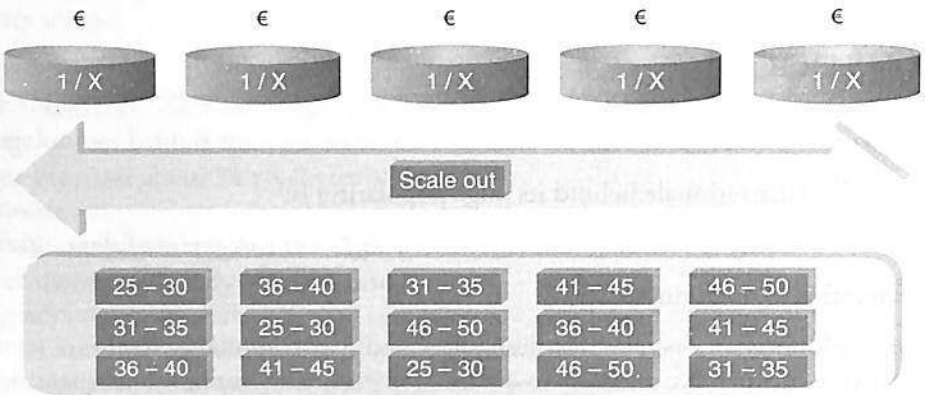


Figure 5.3 Hadoop framework (distributed file system, commodity hardware).

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.

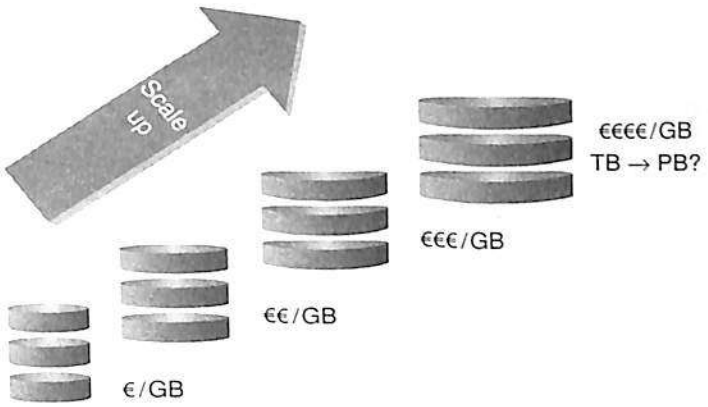


Figure 5.4 RDBMS with respect to cost/GB of storage.

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

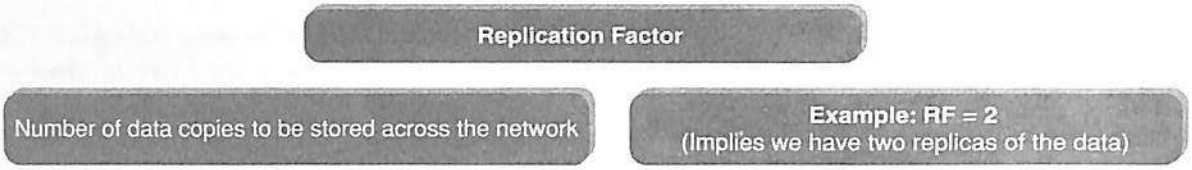


Figure 5.5 Replication factor.

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*".

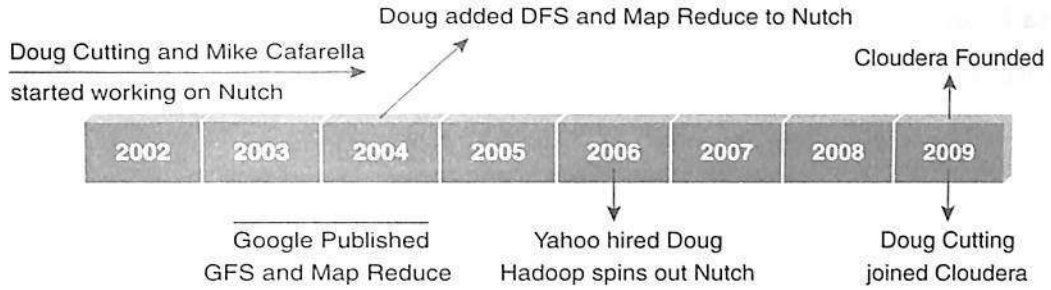


Figure 5.6 Hadoop history.

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.

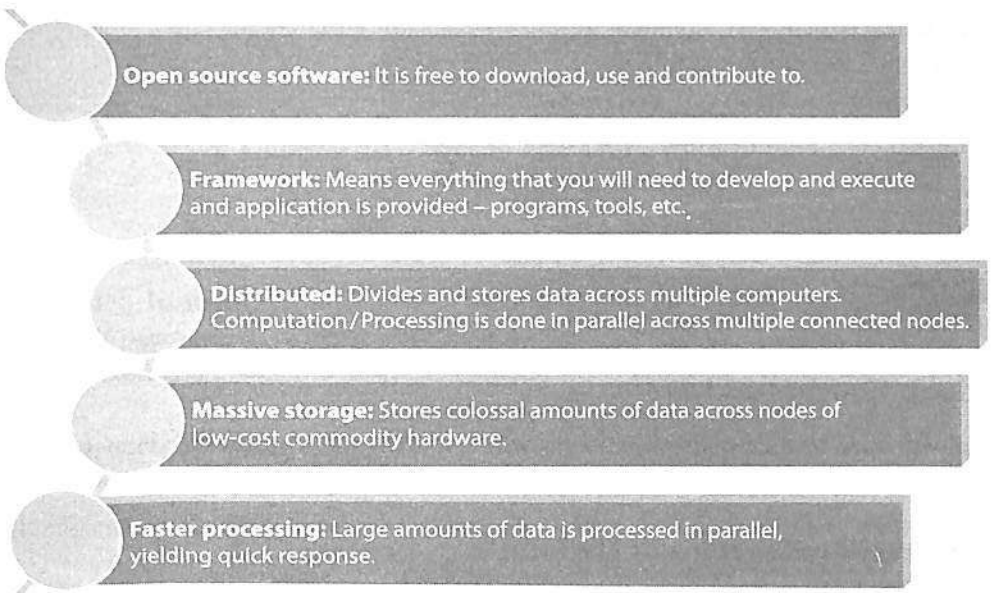


Figure 5.7 Key aspects of Hadoop.

5.7.2 Hadoop Components

Figure 5.8 depicts the Hadoop components.

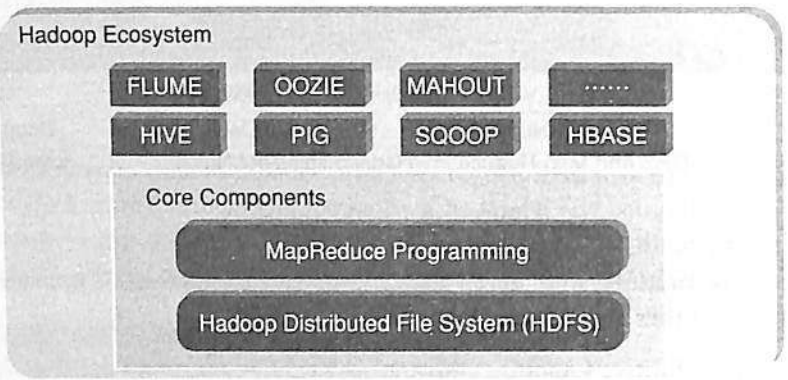


Figure 5.8 Hadoop components.

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.

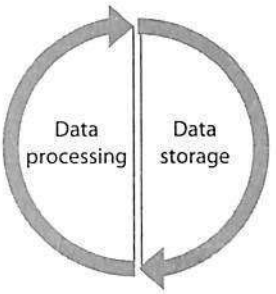


Figure 5.9 Hadoop conceptual layer.

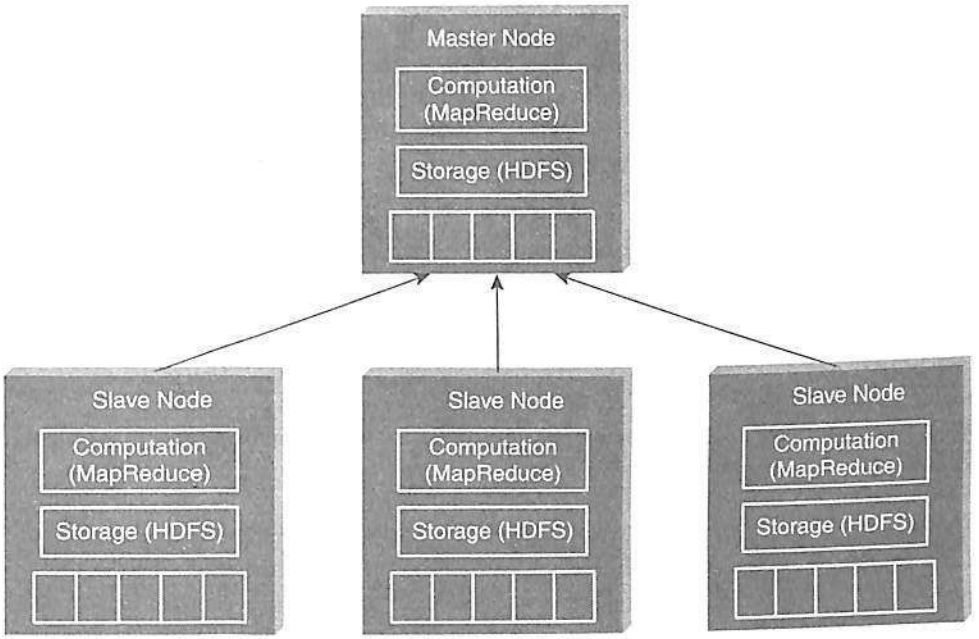


Figure 5.10 Hadoop high-level architecture.
Reference: Hadoop in Practice, Alex Holmes.

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.

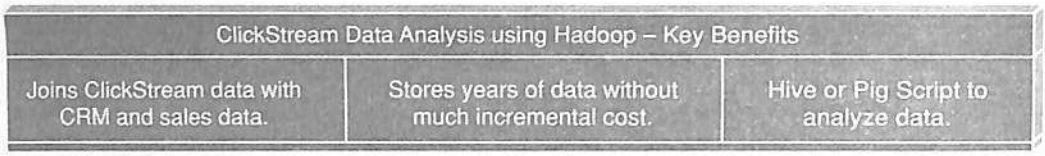


Figure 5.11 ClickStream data analysis.

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.

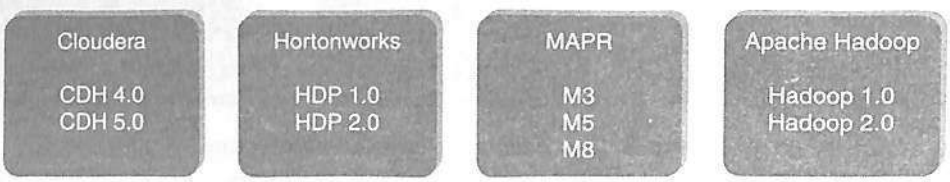


Figure 5.12 Common Hadoop distributors.

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

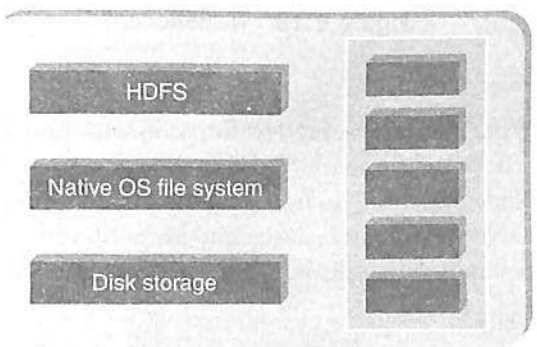


Figure 5.13 Hadoop Distributed File System.

Figure 5.14 Hadoop Distributed File System – key points.

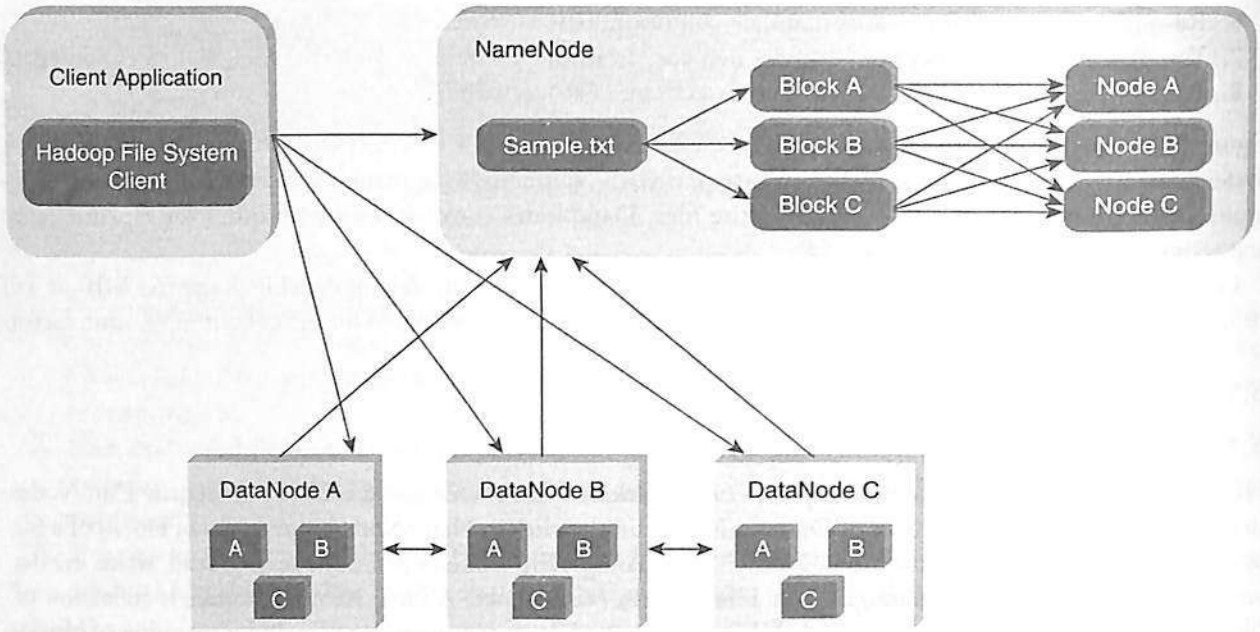


Figure 5.15 Hadoop Distributed File System Architecture.
Reference: Hadoop in Practice, Alex Holmes.

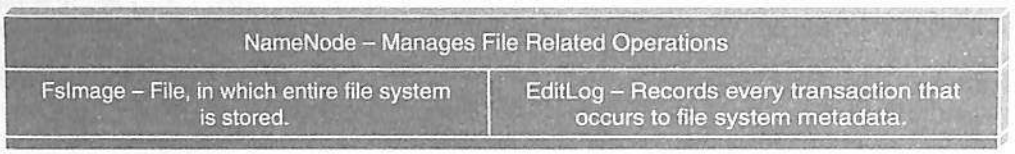


Figure 5.16 NameNode.

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.

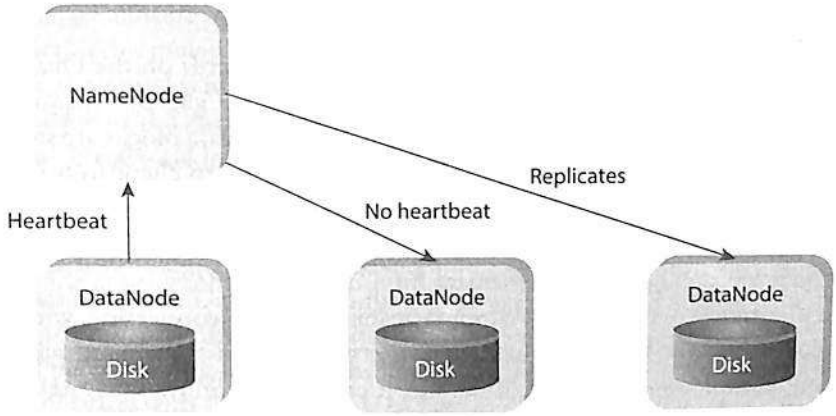


Figure 5.17 NameNode and DataNode Communication.

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.

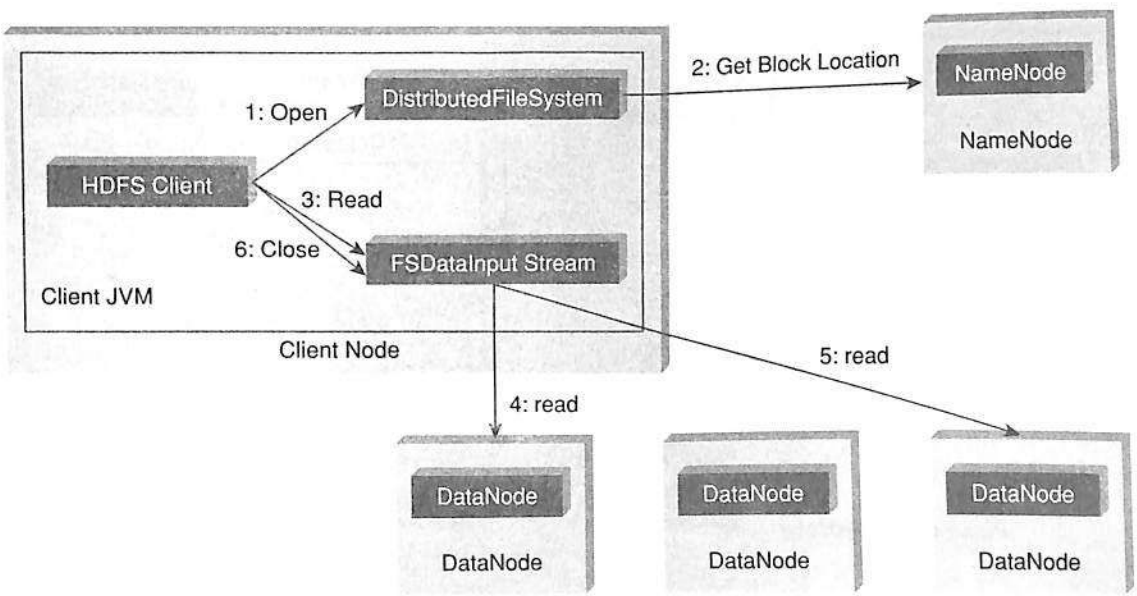


Figure 5.18 File Read.

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

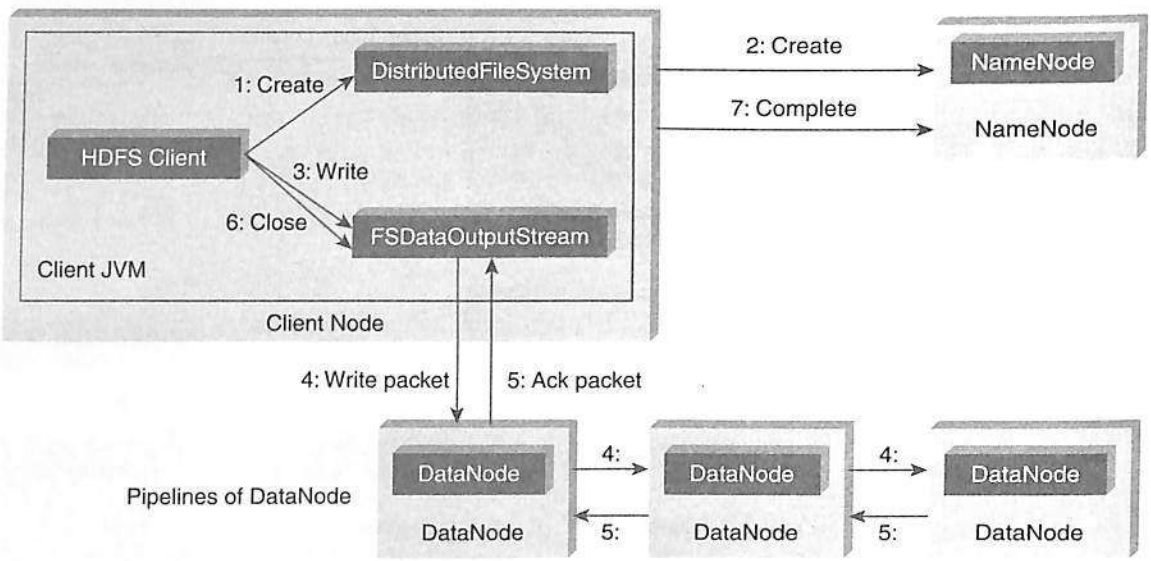


Figure 5.19 File Write.

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 /

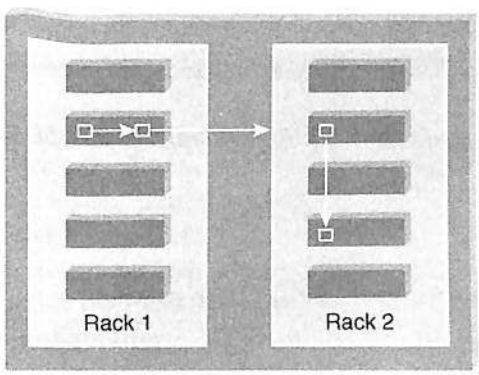


Figure 5.20 Replica Placement Strategy.

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.

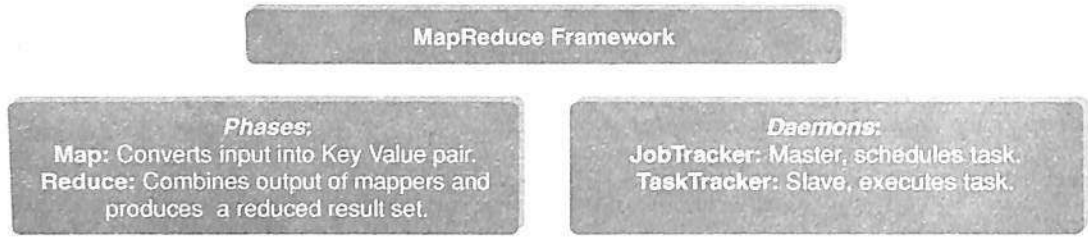


Figure 5.21 MapReduce Programming phases and daemons.

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

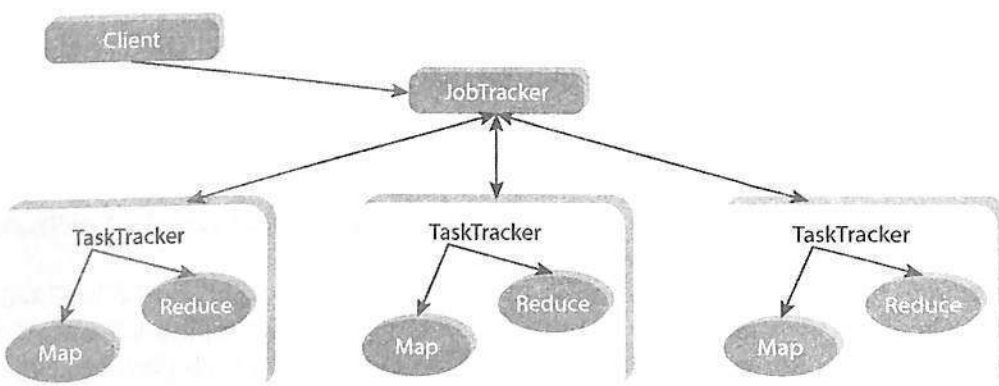


Figure 5.22 JobTracker and TaskTracker interaction.

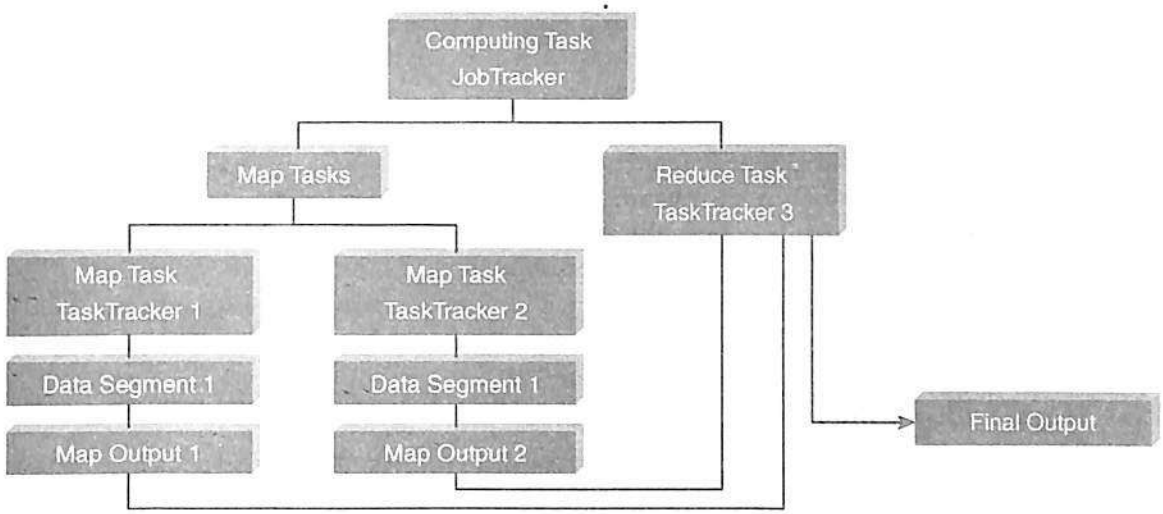


Figure 5.23 MapReduce programming workflow.

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.

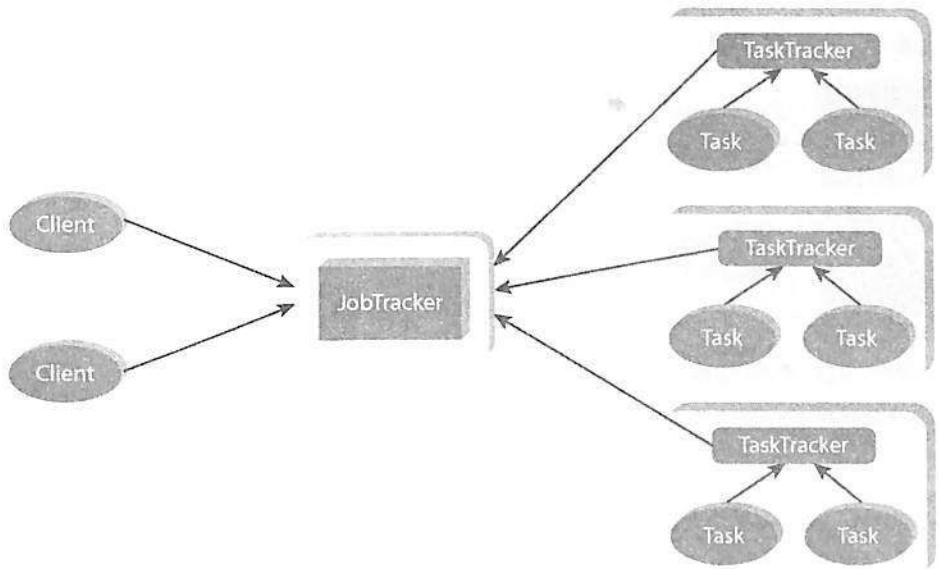


Figure 5.24 MapReduce programming architecture.

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;
```

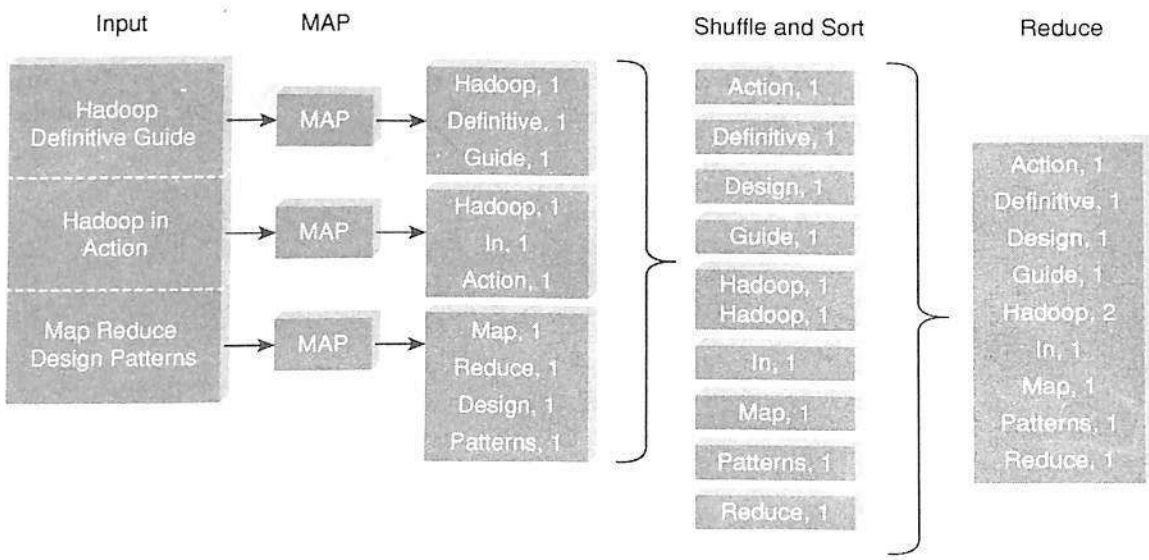


Figure 5.25 Wordcount example.

```
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.

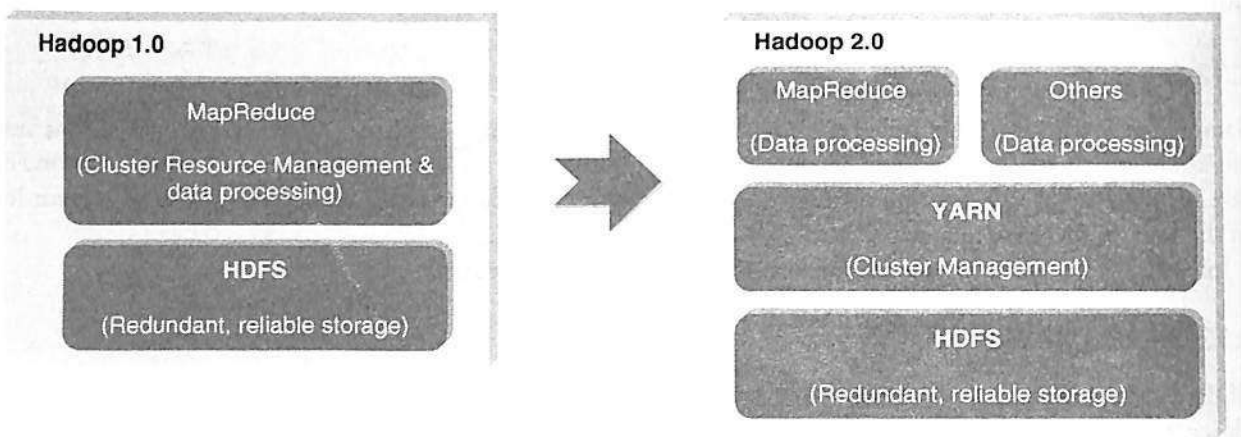


Figure 5.27 Hadoop 1.x versus Hadoop 2.x.

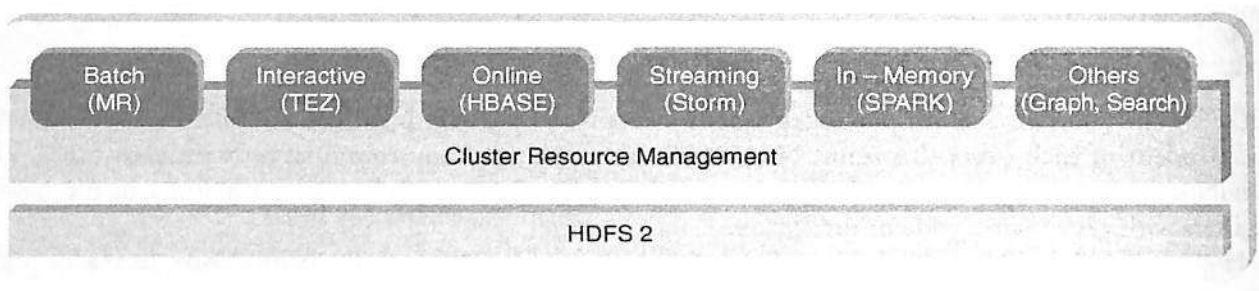


Figure 5.28 Hadoop YARN.

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.

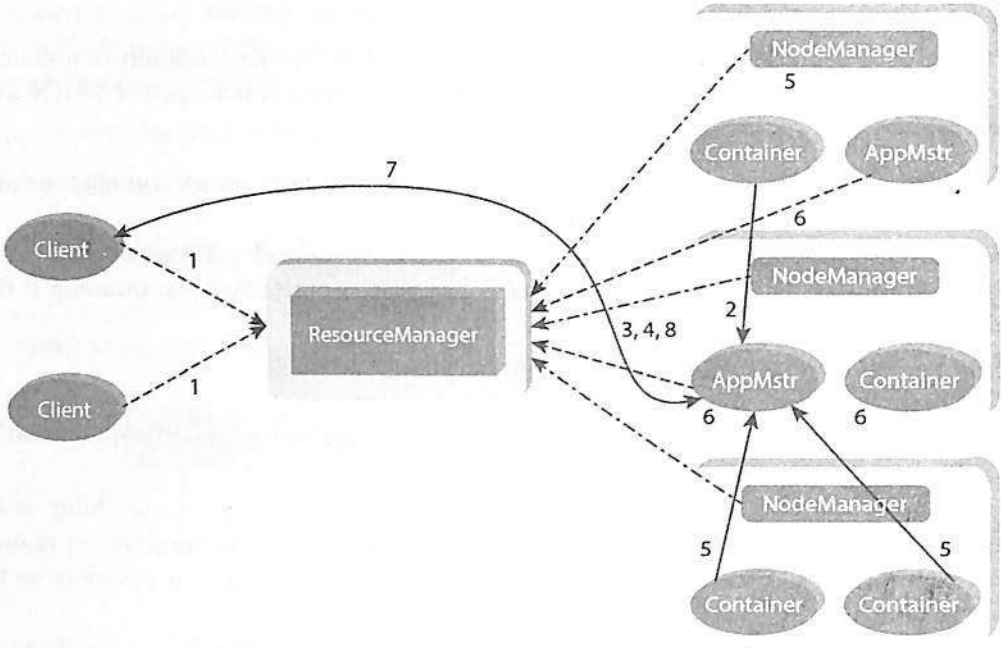


Figure 5.29 YARN architecture.

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.

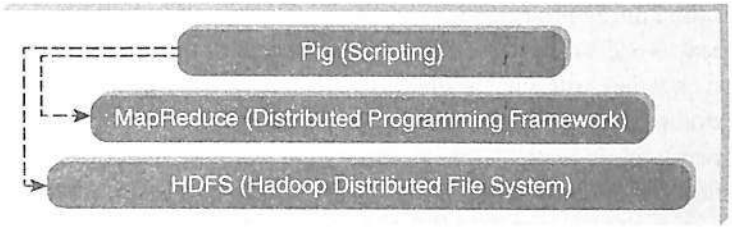


Figure 5.30 Pig in the Hadoop ecosystem.

Figure 5.31 Hive in the Hadoop ecosystem.

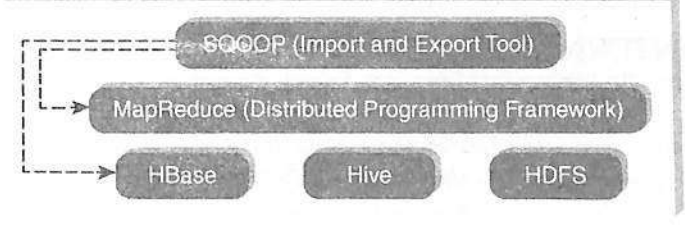


Figure 5.32 Sqoop in the Hadoop ecosystem.

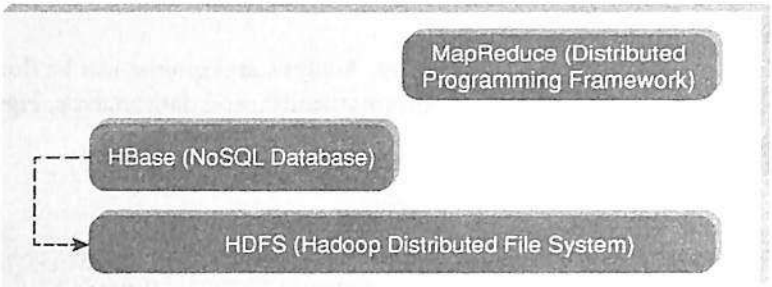


Figure 5.33 HBase in the Hadoop Ecosystem.

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

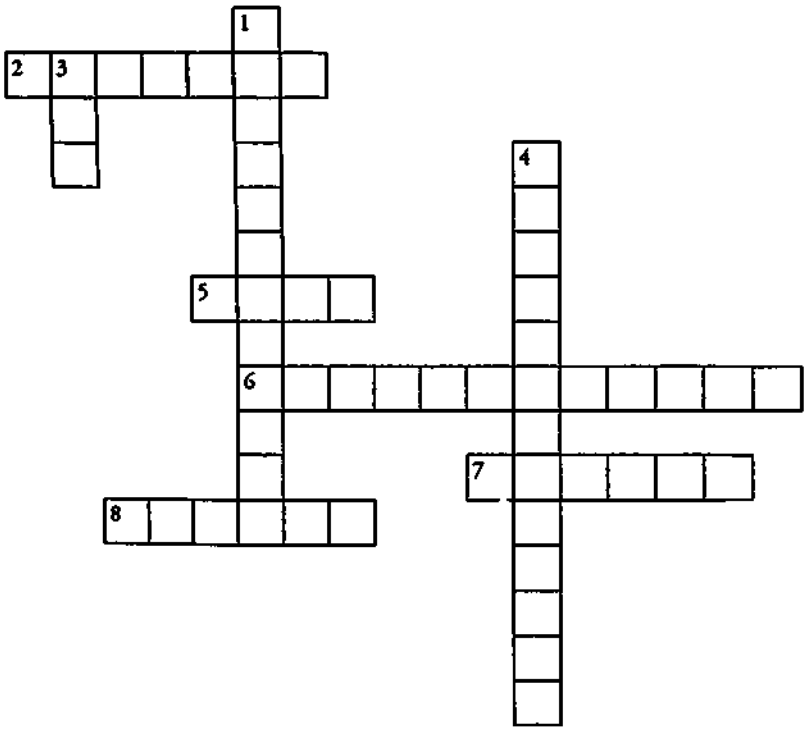
1. Which of the two are components of Hadoop?
   (a) HDFS
   (b) MapReduce
   (c) Secondary NameNode
   (d) Shuffler
   (e) Sqoop
2. How many blocks will be created for a file that is 300 MB? The default block size is 64 MB and the replication factor is 3.
   (a) 30
   (b) 5
   (c) 15
   (d) 100
3. Pig 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",

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

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

6.1 WHAT IS MONGODB?

MongoDB is

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

6.2 WHY MONGODB?

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

6.2.1 Using Java Script Object Notation (JSON)

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

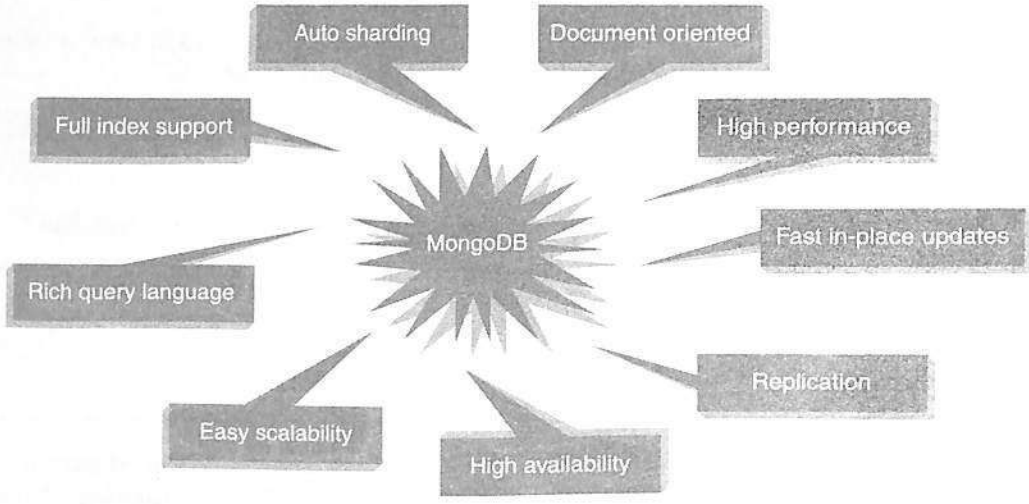


Figure 6.1 Why MongoDB?

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

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

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

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

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

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

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

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

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

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

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

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

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

6.2.2 Creating or Generating a Unique Key

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

0	1	2	3	4	5	6	7	8	9	10	11
Timestamp				Machine ID			Process ID		Counter		

6.2.2.1 Database

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

6.2.2.2 Collection

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

6.2.2.3 Document

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

6.2.3 Support for Dynamic Queries

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

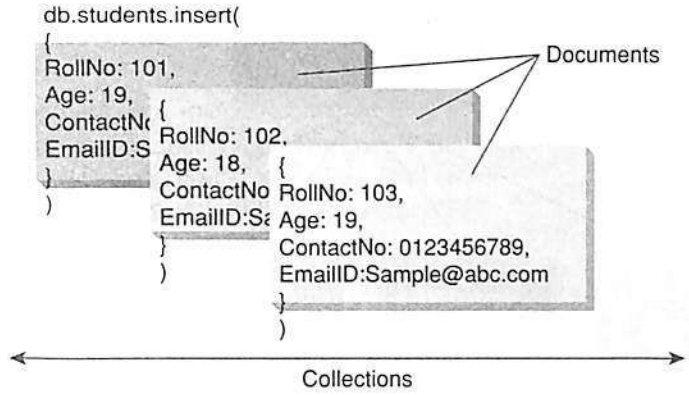


Figure 6.2 A collection "students" containing 3 documents.

6.2.4 Storing Binary Data

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

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

6.2.5 Replication

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

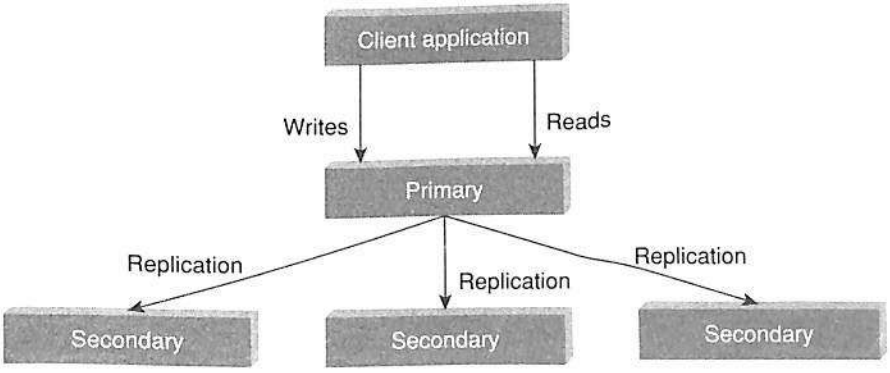


Figure 6.3 The process of **REPLICATION** in MongoDB.

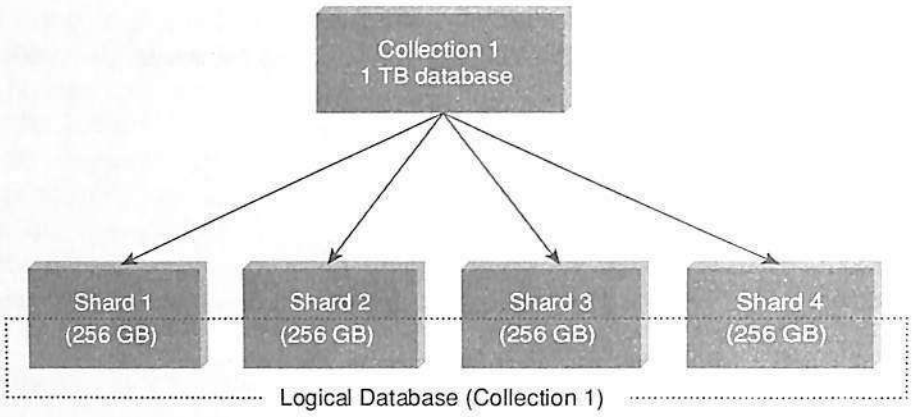


Figure 6.4 The process of **SHARDING** in MongoDB.

6.2.6 Sharding

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

The prime advantages of sharding are as follows:

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

6.2.7 Updating Information In-Place

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

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

Guess Me

A. Who am I?

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

B. Who am I?

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

C. Who am I?

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

D. Who am I?

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

Answers:

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

6.3 TERMS USED IN RDBMS AND MONGODB

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

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

6.3.1 Create Database

The syntax for creating database is as follows:

```
use DATABASE_Name
```

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

```
use myDB
```

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

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

```
db
```

```
> db;
myDB
>
```

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

```
show dbs
```

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

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

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

6.3.2 Drop Database

The syntax to drop database is as follows:

```
db.dropDatabase();
```

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

```
use myDB;
db.dropDatabase();
```

Confirm if the database "myDB" has been dropped.

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

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

6.4 DATA TYPES IN MONGODB

The following are various data types in MongoDB.

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

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

To report the name of the current database:

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

To display the list of databases:

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

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

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

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

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

To display the current version of the MongoDB server:

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

To display the statistics that reflect the use state of a database:

```
C:\windows\system32\cmd.exe - mongo
> db.stats()
{
        "db" : "myDB1",
        "collections" : 3,
        "objects" : 6,
        "avgObjSize" : 122.66666666666667,
        "dataSize" : 736,
        "storageSize" : 24576,
        "numExtents" : 3,
        "indexes" : 1,
        "indexSize" : 8176,
        "fileSize" : 67108864,
        "nsSizeMB" : 16,
        "dataFileVersion" : {
                "major" : 4,
                "minor" : 5
        },
        "extentFreeList" : {
                "num" : 14,
                "totalSize" : 974848
        },
        "ok" : 1
}
>
```

Type in db.help() in the MongoDB client to get a list of commands:

```
C:\windows\system32\cmd.exe - mongo
> db.help();
DB methods:
        db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs comma
nd [ just calls db.runCommand(...) ]
        db.auth(username, password)
        db.cloneDatabase(fromhost)
        db.commandHelp(name) returns the help for the command
        db.copyDatabase(fromdb, todb, fromhost)
        db.createCollection(name, { size : ..., capped : ..., max : ... } )
        db.createUser(userDocument)
        db.currentOp() displays currently executing operations in the db
        db.dropDatabase()
        db.eval(func, args) run code server-side
        db.fsyncLock() flush data to disk and lock server for backups
        db.fsyncUnlock() unlocks server following a db.fsyncLock()
        db.getCollection(cname) same as db['cname'] or db.cname
        db.getCollectionNames()
        db.getLastError() - just returns the err msg string
        db.getLastErrorObj() - return full status object
        db.getMongo() get the server connection object
        db.getMongo().setSlaveOk() allow queries on a replication slave server
        db.getName()
        db.getPrevError()
        db.getProfilingLevel() - deprecated
        db.getProfilingStatus() - returns if profiling is on and slow threshold
        db.getReplicationInfo()
        db.getSiblingDB(name) get the db at the same server as this one
        db.getWriteConcern() - returns the write concern used for any operations
 on this db, inherited from server object if set
        db.hostInfo() get details about the server's host
        db.isMaster() check replica primary status
        db.killOp(opid) kills the current operation in the db
        db.listCommands() lists all the db commands
        db.loadServerScripts() loads all the scripts in db.system.js
        db.logout()
        db.printCollectionStats()
        db.printReplicationInfo()
        db.printShardingStatus()
        db.printSlaveReplicationInfo()
        db.dropUser(username)
        db.repairDatabase()
        db.resetError()
        db.runCommand(cmdObj) run a database command.  if cmdObj is a string, tu
rns it into { cmdObj : 1 }
        db.serverStatus()
        db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all
        db.setWriteConcern( <write concern doc> ) - sets the write concern for w
rites to the db
        db.unsetWriteConcern( <write concern doc> ) - unsets the write concern f
or writes to the db
        db.setVerboseShell(flag) display extra information in shell output
        db.shutdownServer()
        db.stats()
        db.version() current version of the server
>
```

Consider a table "Students" with the following columns:

1. StudRollNo
2. StudName
3. Grade
4. Hobbies
5. DOJ

Before we get into the details of CRUD operations in MongoDB, let us look at how the statements are written in RDBMS and MongoDB.

	RDBMS	MongoDB
Insert	Insert into Students (StudRollNo, StudName, Grade, Hobbies, DOJ) Values ('S101', 'Simon David', 'VII', 'Net Surfing', '10-Oct-2012')	db.Students.insert({_id:1, StudRollNo: 'S101', StudName: 'Simon David', Grade: 'VII', Hobbies: 'Net Surfing', DOJ: '10-Oct-2012'});
Update	Update Students set Hobbies ='Ice Hockey' where StudRollNo ='S101'	db.Students.update({StudRollNo: 'S101'}, {$set: {Hobbies : 'Ice Hockey'}})
	Update Students Set Hobbies ='Ice Hockey'	db.Students.update({},{$set: {Hobbies: 'Ice Hockey' }}, {multi:true})
Delete	Delete from Students where StudRollNo = 'S101'	db.Students.remove ({StudRollNo : 'S101'})
	Delete From Students	db.Students.remove({})
Select	Select * from Students	db.Students.find() db.Students.find().pretty()
	Select * from students where StudRollNo = 'S101'	db.Students.find({StudRollNo: 'S101'})
	Select StudRollNo, StudName, Hobbies from Students	db.Students.find({},{StudRollNo:1, StudName:1, Hobbies:1, _id:0})
	Select StudRollNo, StudName, Hobbies from Students where StudRollNo = 'S101'	db.Students.find({StudRollNo: 'S101'}, {StudRollNo : 1, StudName: 1, Hobbies : 1, _id:0})

RDBMS	MongoDB
Select StudRollNo, StudName, Hobbies From Students Where Grade ='VII' and Hobbies ='Ice Hockey'	db.Students.find({Grade: 'VII' , Hobbies: 'Ice Hockey'}, {StudRollNo : 1, StudName: 1, Hobbies : 1, _id:0})
Select StudRollNo, StudName, Hobbies From Students Where Grade ='VII' or Hobbies = 'Ice Hockey'	db.Students.find({ $or: [{Grade: 'VII' , Hobbies: 'Ice Hockey'}], StudRollNo : 1, StudName: 1, Hobbies : 1, _id:0})
Select * From Students Where StudName like 'S%'	db.Students.find({ StudName: /^S/}).pretty()

6.5 MONGODB QUERY LANGUAGE

CRUD (Create, Read Update, and Delete) operations in MongoDB

Create → Creation of data is done using insert() or update() or save() method.
Read → Reading the data is performed using the find() method.
Update → Update to data is accomplished using the update() method with UPSERT set to false.
Delete → a document is Deleted using the remove() method.

We will present the various methods available in MongoDB shell to deal with data in the next few sections. 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.

At few places we have also provided the equivalent in RDBMS such as Oracle.

Objective: To create a collection by the name "Person". Let us take a look at the collection list prior to the creation of the new collection "Person":

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

Act: The statement to create the collection is
db.createCollection("Person")

```
C:\windows\system32\cmd.exe - mongo
> db.createCollection("Person");
{ "ok" : 1 }
>
```

Outcome: Below is the collection list after the creation of the new collection "Person":

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

Objective: To drop a collection by the name "food".
Take a look at the current collection list:

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

Act: The statement to drop the collection is

db.food.drop();

```
C:\windows\system32\cmd.exe - mongo
> db.food.drop();
true
>
```

Outcome: The collection list after the execution of the statement is as follows:

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

6.5.1 Insert Method

We now explain the syntax of insert method.

```
db.students.insert(          <-------  Collection
{
RollNo: 101,                 <-------  Field: value
Age: 19,                     <-------  Field: value
ContactNo:0123456789,        <-------  Field: value
EmailID:Sample@abc.com       <-------  Field: value
}
)
```

Objective: To create a collection by the name "Students" and insert documents.

Input: Check the list of existing collections.

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

Act: Create a collection by the name "Students" and store the following data in it.

db.Students.insert({_id:1, StudName:"Michelle Jacintha", Grade: "VII", Hobbies: "Internet Surfing"});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.insert({_id:1, StudName:"Michelle Jacintha", Grade: "VII", Hobbies: "Internet Surfing"});
WriteResult({ "nInserted" : 1 })
>
```

Outcome: Check if the collection has been successfully created.

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

Check if the document for Student "Michelle Jacintha" has been successfully inserted into the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find();
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
>
```

To format the result, one can add the pretty() method to the operation.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

Objective: Insert another document into the collection.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

Act:

db.Students.insert({_id:2, StudName:"Mabel Mathews", Grade: "VII", Hobbies: "Baseball"});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.insert({_id:2, StudName:"Mabel Mathews", Grade: "VII", Hobbies: "Baseball"});
WriteResult({ "nInserted" : 1 })
>
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Objective: Insert the document for "Aryan David" into the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from "Skating" to "Chess".) Use "**Update else insert**" (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it).

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Act:

db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Skating"}},{upsert:true});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Skating"}},{upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
>
```

Outcome: Confirm the presence of the document of "Aryan David" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:3});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Skating" }
>
```

Objective: Insert the document for "Aryan David" into the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from "Skating" to "Chess"). Use "**Update else insert**" (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it). Try the UPSERT operator by setting it first to "true" and then to "false" and observe the output.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Skating"
}
>
```

Act:

db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Chess"}},{upsert:true});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:3, StudName:"Aryan David", Grade: "VII"},{$set:{Hobbies: "Chess"}},{upsert:true});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Confirm that the required changes have been made to the document of "Aryan David" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:3});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
>
```

Objective: To demonstrate Save method to insert a document for student "Vamsi Bapat" in the "Students" collection. Omit providing value for the _id key.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
>
```

Act:

db.Students.save({StudName:"Vamsi Bapat",Grade:"VI"})

```
C:\windows\system32\cmd.exe - mongo
> db.Students.save({StudName:"Vamsi Bapat",Grade:"VI"})
WriteResult({ "nInserted" : 1 })
>
```

Outcome: Confirm the presence of the document of "Vamsi Bapat" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find().pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : ObjectId("546dd0e0a7fba710799bb94d"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

6.5.2 Save() Method

We now explain the save() method. The save() method will insert a new document if the document with the specified _id does not exist. However, if a document with the specified id exists, it replaces the existing document with the new one.

Objective: Insert the document of "Hersch Gibbs" into the Students collection using the Update method. First try with upsert set to false and then with upsert set to true.

Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find();
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
{ "_id" : ObjectId("546dd0e0a7fba710799bb94d"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
>
```

Act: *Update method with upsert set to false.*

db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:false});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:false});
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })
>
```

As evident from the above display (nUpserted : 0), no document has been inserted because upsert is set to false.

Update method with upsert set to true.

db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:true});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4, StudName:"Hersch Gibbs", Grade: "VII"},{$set:{Hobbies: "Graffiti"}}, {upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 4 })
>
```

nUpserted: 1 implies that a document with _id:4 has been inserted.

Outcome: Confirm the presence of the document of "Hersch Gibbs" in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find();
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
{ "_id" : ObjectId("546dd0e0a7fba710799bb94d"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti" }
>
```

6.5.3 Adding a New Field to an Existing Document – Update Method

We now discuss the syntax of update method.

```
db.students.update(      <-------  Collection
{Age: {$gt 18}},         <-------  Update Criteria
{$set: {Status: "A" }}   <-------  Update Action
{multi:true}             <-------  Update Option
)
```

Objective: To add a new field "Location" with value "Newark" to the document (_id:4) of "Students" collection.

Input: Check the document (_id:4) in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

Act:

db.Students.update({_id:4},{$set:{Location:"Newark"}});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4},{$set:{Location:"Newark"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Confirm that the new field "Location" with value "Newark" has been added to document (_id:4) in the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti",
        "Location" : "Newark"
}
>
```

6.5.4 Removing an Existing Field from an Existing Document – Remove Method

In this section we will explain the syntax of remove method.

```
db.students.remove(      <----------  Collection
{Age: {$gt 18}},         <----------  Remove Criteria
)
```

Objective: To remove the field "Location" with value "Newark" in the document (_id:4) of "Students" collection.

Input: Check the document (_id:4) in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti",
        "Location" : "Newark"
}
>
```

Act:

db.Students.update({_id:4},{$unset:{Location:"Newark"}});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:4},{$unset:{Location:"Newark"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Confirm if the stated field ("Location") has been dropped from the document (_id:4) of the "Students" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:4}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

6.5.5 Finding Documents based on Search Criteria – Find Method

The syntax of find method is as follows:

```
db.students.find(          <----------  Collection
{Age: {$gt 18}},           <----------  Selection Criteria
{RollNo:1,Age:1,_id:1}     <----------  Projection
).limit(10)                <----------  Cursor Modifier
```

Objective: To search for documents from the "Students" collection based on certain search criteria.
Input: Check the documents in the "Students" collection before proceeding.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Act: Find the document wherein the "StudName" has value "Aryan David".

db.Students.find({StudName:"Aryan David"});

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:"Aryan David"});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
>
```

To format the above output, use the pretty() method:

db.Students.find({StudName:"Aryan David"}).pretty();

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:"Aryan David"}).pretty();
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
>
```

RDBMS equivalent:

```
Select *
    From Students
        Where StudName like 'Aryan David';
```

```
SQL> select * from Students where StudName like 'Aryan David';

STUDR STUDNAME                    GRADE HOBBIES
----- -------------------------- ----- ------------------------------
3     Aryan David                VII   Chess

SQL>
```

Objective: To display only the StudName from all the documents of the Student's collection. The identifier "_id" should be suppressed and NOT displayed.

Act:

db.Students.find({},{StudName:1,_id:0});

Outcome:

```
> db.Students.find({},{StudName:1,_id:0});
{ "StudName" : "Michelle Jacintha" }
{ "StudName" : "Aryan David" }
{ "StudName" : "Hersch Gibbs" }
{ "StudName" : "Vamsi Bapat" }
{ "StudName" : "Mabel Mathews" }
>
```

RDBMS equivalent:

Select StudName
From Students;

```
SQL> select StudName from Students;

STUDNAME
-------------------------
Michelle Jacintha
Aryan David
Mabel Mathews
Hersch Gibbs
Vamsi Bapat

SQL>
```

Objective: To display only the StudName and Grade from all the documents of the Students collection. The identifier _id should be suppressed and NOT displayed.

Act:

db.Students.find({},{StudName:1,Grade:1,_id:0});

Outcome:

```
> db.Students.find({},{StudName:1,Grade:1,_id:0});
{ "StudName" : "Michelle Jacintha", "Grade" : "VII" }
{ "Grade" : "VII", "StudName" : "Aryan David" }
{ "Grade" : "VII", "StudName" : "Hersch Gibbs" }
{ "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "StudName" : "Mabel Mathews", "Grade" : "VII" }
>
```

RDBMS equivalent:

Select StudName, Grade
From Students;

```
SQL> select StudName, Grade from Students;

STUDNAME                  GRADE
------------------------- -----
Michelle Jacintha         VII
Aryan David               VII
Mabel Mathews             VII
Hersch Gibbs              VII
Vamsi Bapat               VI

SQL>
```

Objective: To display the StudName, Grade as well the identifier, _id from the document of the Students collection where the _id column is 1.

Act:

db.Students.find({_id:1},{StudName:1,Grade:1});

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:1},{StudName:1,Grade:1});
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII" }
>
```

RDBMS equivalent:

```
Select StudRollNo, StudName, Grade
        From Students
                Where StudRollNo = '1';
```

```
SQL Plus
SQL> select StudRollNo, StudName, Grade from Students where StudRollNo = '1';
STUDR STUDNAME                 GRADE
----- ---------------------- -------
1     Michelle Jacintha        VII
SQL>
```

Objective: To display the StudName and Grade from the document of the Students collection where the _id column is 1. The _id field should NOT be displayed.

Act:

```
db.Students.find({_id:1},{StudName:1,Grade:1,_id:0});
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({_id:1},{StudName:1,Grade:1,_id:0});
{ "StudName" : "Michelle Jacintha", "Grade" : "VII" }
>
```

RDBMS equivalent:

```
Select StudName, Grade
        From Students
                Where StudRollNo like '1';
```

```
SQL Plus
SQL> select StudName, Grade from Students where StudRollNo like '1';
STUDNAME                 GRADE
------------------------ -------
Michelle Jacintha        VII
SQL>
```

Relational operators available to use in the search criteria:

$eq	→ equal to
$ne	→ not equal to
$gte	→ greater than or equal to
$lte	→ less than or equal to
$gt	→ greater than
$lt	→ less than

Objective: To find those documents where the Grade is set to 'VII'.

Act:

```
db.Students.find({Grade:{$eq:'VII'}}).pretty();
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({Grade:{$eq:'VII'}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where Grade like 'VII';

```
SQL Plus
SQL> select * from Students where Grade like 'VII';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- --------------------------------
1     Michelle Jacintha           VII   Internet surfing
3     Aryan David                 VII   Chess
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti

SQL>
```

Objective: To find those documents where the Grade is NOT set to 'VII'.

Act:

db.Students.find({Grade:{$ne:'VII'}}).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({Grade:{$ne:'VII'}}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

There is just one document that meets the above criteria of Grade NOT EQUAL to 'VII'.

RDBMS Equivalent:

Select *

From Students

Where Grade <> 'VII';

```
SQL Plus
SQL> select * from Students where Grade <> 'VII';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- --------------------------------
      Vamsi Bapat                 VI

SQL>
```

OR

```
SQL Plus
SQL> select * from Students where Grade != 'VII';
STUDR STUDNAME                    GRADE HOBBIES
----- -------------------------- ----- --------------------------------
      Vamsi Bapat                VI
SQL>
```

Objective: To find those documents from the Students collection where the Hobbies is set to either 'Chess' or is set to 'Skating'.

Act:

db.Students.find ({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find ({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where Hobbies in ('Chess', 'Skating');

```
SQL Plus
SQL> select * from Students where Hobbies in ('Chess', 'Skating');
STUDR STUDNAME                    GRADE HOBBIES
----- -------------------------- ----- --------------------------------
3     Aryan David                VII   Chess
SQL>
```

Objective: To find those documents from the Students collection where the Hobbies is set neither to 'Chess' nor is set to 'Skating'.

Act:

db.Students.find ({Hobbies :{ $nin: ['Chess','Skating']}}).pretty ();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find ({Hobbies :{ $nin: ['Chess','Skating']}}).pretty ();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where Hobbies not in ('Chess', 'Skating');
```

```
SQL> select * from Students where Hobbies not in ('Chess','Skating');

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
SQL>
```

Objective: To find those documents from the Students collection where the Hobbies is set to 'Graffiti' and the StudName is set to 'Hersch Gibbs' (AND condition).

Act:

```
db.Students.find({Hobbies:'Graffiti', StudName: 'Hersch Gibbs'}).pretty();
```

Outcome:

```
> db.Students.find({Hobbies:'Graffiti', StudName: 'Hersch Gibbs'}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where Hobbies like 'Graffiti' and StudName like 'Hersch Gibbs';
```

```
SQL> select * from Students where Hobbies like 'Graffiti' and StudName like 'Her
sch Gibbs';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
4     Hersch Gibbs                VII   Graffiti
SQL>
```

Objective: To find documents from the Students collection where the StudName begins with "M".

Act:

```
db.Students.find({StudName:/^M/}).pretty();
```

Outcome:

```
> db.Students.find({StudName:/^M/}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where StudName like 'M%';
```

```
SQL Plus

SQL> select * from Students where StudName like 'M%';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball

SQL>
```

Objective: To find documents from the Students collection where the StudName ends in "s".

Act:

```
db.Students.find({StudName:/s$/}).pretty();
```

Outcome:

```
C:\Windows\system32\cmd.exe - mongo
> db.Students.find({StudName:/s$/}).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select *
        From Students
                Where StudName like '%s';
```

```
SQL Plus

SQL> select * from Students where StudName like '%s';

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti

SQL>
```

Objective: To find documents from the Students collection where the StudName has an "e" in any position.

Act:

```
db.Students.find({StudName:/e/}).pretty();
```

OR

```
db.Students.find({StudName:/.*e.*/}).pretty();
```

OR

```
db.Students.find({StudName:{$regex:"e"}}).pretty();
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:/e/}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where StudName like '%e%';

```
SQL Plus
SQL> select * from Students where StudName like '%e%';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
SQL>
```

Objective: To find documents from the Students collection where the StudName ends in "a".

Act:

db.Students.find({StudName:{$regex:"a$"}}).pretty();

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({StudName:{$regex:"a$"}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
>
```

RDBMS Equivalent:

Select *

From Students

Where StudName like '%a';

```
SQL Plus
SQL> select * from Students where StudName like '%a';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
SQL>
```

Objective: To find documents from the Students collection where the StudName begins with "M".

Act:

```
db.Students.find({StudName:{$regex:"^M"}}).pretty();
```

Outcome:

```
> db.Students.find({StudName:{$regex:"^M"}}).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
```

RDBMS Equivalent:

```
Select *
        From Students
                Where StudName like 'M%';
```

```
SQL> select * from Students where StudName like 'M%';
STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ------------------------------
1     Michelle Jacintha           VII   Internet surfing
2     Mabel Mathews               VII   Baseball
SQL>
```

6.5.6 Dealing with NULL Values

Objective: To add a new field with null value in existing documents (_id:3 and _id:4) of Students collection. A NULL is a missing or unknown value. When we place NULL as a value for a field, it implies that currently we do not know the value or the value is missing. We can always update the value of the field once we know it.

Input: Before we execute the commands to update documents with a null value in a column, let us first view the two documents.

```
db.Students.find({$or:[{_id:3},{_id:4}]})
```

```
> db.Students.find({$or:[{_id:3},{_id:4}]});
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess" }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti" }
>
```

Act: Update the documents with NULL values in the "Location" column.

```
db.Students.update({_id:3},{$set:{Location:null}});
db.Students.update({_id:4},{$set:{Location:null}});
```

```
> db.Students.update({_id:3},{$set:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Students.update({_id:4},{$set:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

RDBMS Equivalent:

Update Students
Set Location = null
Where StudRollNo in ('3','4');

```
SQL Plus
SQL> update Students set Location = null where StudRollNo in ('3','4');
2 rows updated.
SQL>
```

Outcome: To search for NULL values in Location column.

db.Students.find({Location:{$eq:null}});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({Location:{$eq:null}});
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess", "Location" : null }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti", "Location" : null }
{ "_id" : ObjectId("5464849889ad1ab07d489b7f"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
>
```

The above statement displays documents which either have NULL values in the location column or do not have the location column at all.

RDBMS Equivalent:

Select *
From Students
Where Location is Null;

```
SQL Plus
SQL> select * from Students where Location is NULL;
STUDR STUDNAME                    GRADE HOBBIES                        LOCATION
----- --------------------------- ----- ------------------------------ ----------
---------
1     Michelle Jacintha           VII   Internet surfing
3     Aryan David                 VII   Chess
2     Mabel Mathews               VII   Baseball
4     Hersch Gibbs                VII   Graffiti
      Vamsi Bapat                 VI

SQL>
```

Objective: To remove "Location" field having "NULL" values from the documents (_id:3 and _id:4) from the Students collection.

Input: Document from the "Students" collection having "NULL" values in the "Location" column.

```
C:\windows\system32\cmd.exe - mongo
> db.Students.find({Location:{$eq:null}});
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" : "Internet Surfing" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess", "Location" : null }
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graffiti", "Location" : null }
{ "_id" : ObjectId("5464849889ad1ab07d489b7f"), "StudName" : "Vamsi Bapat", "Grade" : "VI" }
{ "_id" : 2, "StudName" : "Mabel Mathews", "Grade" : "VII", "Hobbies" : "Baseball" }
>
```

Act:

db.Students.update({_id:3},{$unset:{Location:null}});
db.Students.update({_id:4},{$unset:{Location:null}});

```
C:\windows\system32\cmd.exe - mongo
> db.Students.update({_id:3},{$unset:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.Students.update({_id:4},{$unset:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
>
```

Outcome: Let us confirm if the changes have been made by running find method on the Students collection.

```
> db.Students.find({})
{ "_id" : 1, "StudName" : "Michelle Jacintha", "Grade" : "VII", "Hobbies" :
{ "_id" : 3, "Grade" : "VII", "StudName" : "Aryan David", "Hobbies" : "Chess
{ "_id" : 4, "Grade" : "VII", "StudName" : "Hersch Gibbs", "Hobbies" : "Graf
{ "_id" : ObjectId("546484988 9ad1ab07d489b7f"), "StudName" : "Vamsi Bapat",
```

6.5.7 Count, Limit, Sort, and Skip

Objective: To find the number of documents in the Students collection.

Act:

```
db.Students.count()
```

Outcome:

```
> db.Students.count()
5
>
```

Objective: To find the number of documents in the Students collection wherein the Grade is VII.

Act:

```
db.Students.count({Grade:"VII"});
```

Outcome:

```
> db.Students.count({Grade:"VII"});
4
>
```

Objective: To retrieve the first 3 documents from the Students collection wherein the Grade is VII.

Act:

```
db.Students.find({Grade:"VII"}).limit(3).pretty();
```

Outcome:

```
> db.Students.find({Grade:"VII"}).limit(3).pretty();
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
>
```

RDBMS Equivalent:

Select *

From Students
Where Grade like 'VII' and rownum < 4;

```
SQL> select * from Students where Grade like 'VII' and rownum < 4;
STUDR STUDNAME                    GRADE HOBBIES                        LOCATION
----- --------------------------- ----- ------------------------------ ----------
    1 Michelle Jacintha           VII   Internet surfing
    3 Aryan David                 VII   Chess
    2 Nabel Mathews               VII   Baseball
SQL>
```

Objective: To sort the documents from the Students collection in the ascending order of StudName.

Act:

```
db.Students.find().sort({StudName:1}).pretty();
```

Outcome:

```
> db.Students.find().sort({StudName:1}).pretty();
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Nabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet surfing"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

RDBMS Equivalent:

Select *
From Students
Order by StudName asc;

```
SQL> select * from Students order by StudName asc;
STUDR STUDNAME                    GRADE HOBBIES                        LOCATION
----- --------------------------- ----- ------------------------------ ----------
    3 Aryan David                 VII   Chess
    4 Hersch Gibbs                VII   Graffiti
    2 Nabel Mathews               VII   Baseball
    1 Michelle Jacintha           VII   Internet surfing
      Vamsi Bapat                 VI
SQL>
```

Objective: To sort the documents from the Students collection in the descending order of StudName.

Act:

```
db.Students.find().sort({StudName:-1}).pretty();
```

Outcome:

```
> db.Students.find().sort({StudName:-1}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
```

RDBMS Equivalent:

Select *

From Students

Order by StudName desc;

```
SQL> select * from Students order by StudName desc;

STUDR STUDNAME                 GRADE HOBBIES                        LOCATION
----- ------------------------ ----- ------------------------------ ----------
      Vamsi Bapat              VI
1     Michelle Jacintha        VII   Internet surfing
2     Mabel Mathews            VII   Baseball
4     Hersch Gibbs             VII   Graffiti
3     Aryan David              VII   Chess

SQL>
```

Objective: To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in descending order.

Act:

db.Students.find().sort({Grade:1, Hobbies:-1}).pretty();

Outcome:

```
> db.Students.find().sort({Grade:1, Hobbies:-1}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
```

RDBMS Equivalent:

```
Select *
        From Students
                Order by Grade asc, hobbies desc;
```

```
SQL> select * from Students order by Grade asc, Hobbies desc;

STUDR STUDNAME                    GRADE HOBBIES                         LOCATION
----- --------------------------- ----- ------------------------------ ----------
      Vamsi Bapat                 VI
1     Michelle Jacintha           VII   Internet surfing
3     Hersch Gibbs                VII   Graffiti
2     Aryan David                 VII   Chess
4     Mabel Mathews               VII   Baseball

SQL>
```

Objective: To sort the documents from the Students collection first on Grade in ascending order and then on Hobbies in ascending order.

Act:

```
db.Students.find().sort({Grade:1, Hobbies:1}).pretty();
```

Outcome:

```
> db.Students.find().sort({Grade:1, Hobbies:1}).pretty();
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 3,
        "Grade" : "VII",
        "StudName" : "Aryan David",
        "Hobbies" : "Chess"
}
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet surfing"
}
```

RDBMS Equivalent:

```
Select *
        From Students
                Order by Grade asc, Hobbies asc;
```

```
SQL> select * from Students order by Grade asc, Hobbies asc;

STUDR STUDNAME                    GRADE HOBBIES                         LOCATION
----- --------------------------- ----- ------------------------------ ----------
      Vamsi Bapat                 VI
4     Mabel Mathews               VII   Baseball
2     Aryan David                 VII   Chess
3     Hersch Gibbs                VII   Graffiti
1     Michelle Jacintha           VII   Internet surfing

SQL>
```

Objective: To skip the first 2 documents from the Students collection.

Act:

```
db.Students.find().skip(2).pretty();
```

Outcome:

```
> db.Students.find().skip(2).pretty();
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

RDBMS Equivalent:

```
Select StudRollNo, StudName, Grade, Hobbies
        From (Select StudRollNo, StudName, Grade, Hobbies, RowNum as TheRowNum
                From Students)
                        Where TheRowNum > 2;
```

```
SQL> Select StudRollNo, StudName, Grade, Hobbies from (Select StudRollNo, StudNa
me, Grade, Hobbies, Rownum as therownum from Students) where theRownum > 2;

STUDR STUDNAME                   GRADE HOBBIES
----- ------------------------- ----- ------------------------------
2     Mabel Mathews             VII   Baseball
4     Hersch Gibbs              VII   Graffiti
      Vamsi Bapat               VI

SQL>
```

Objective: To sort the documents from the Students collection and skip the first document from the output.

Act:

```
db.Students.find().skip(1).pretty().sort({StudName:1});
```

Outcome:

```
> db.Students.find().skip(1).pretty().sort({StudName:1});
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
{
        "_id" : 1,
        "StudName" : "Michelle Jacintha",
        "Grade" : "VII",
        "Hobbies" : "Internet Surfing"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
>
```

RDBMS Equivalent:

```
Select StudRollNo, StudName, Grade, Hobbies
    From (Select StudRollNo, StudName, Grade, Hobbies, RowNum as TheRowNum
        From Students)
            Where TheRowNum > 1
                Order by StudName;
```

```
SQL> Select StudRollNo, StudName, Grade, Hobbies from (Select StudRollNo, StudNa
me, Grade, Hobbies, RowNum as therownum from Students) where therownum >1 order
by StudName;

STUDR STUDNAME                    GRADE HOBBIES
----- --------------------------- ----- ---------------------------------------
    3 Aryan David                 VII   Chess
    4 Hersch Gibbs                VII   Graffiti
    2 Mabel Mathews               VII   Baseball
    1 Vamsi Bapat                 VI

SQL>
```

Objective: To display the last 2 records from the Students collection.

Act:

```
db.Students.find().pretty().skip(db.Students.count()-2);
```

Outcome:

```
> db.Students.find().pretty().skip(db.Students.count()-2);
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

Objective: To retrieve the third, fourth, and fifth document from the Students collection.

Act:

```
db.Students.find().pretty().skip(2).limit(3);
```

Outcome:

```
> db.Students.find().pretty().skip(2).limit(3);
{
        "_id" : 4,
        "Grade" : "VII",
        "StudName" : "Hersch Gibbs",
        "Hobbies" : "Graffiti"
}
{
        "_id" : ObjectId("5464849889ad1ab07d489b7f"),
        "StudName" : "Vamsi Bapat",
        "Grade" : "VI"
}
{
        "_id" : 2,
        "StudName" : "Mabel Mathews",
        "Grade" : "VII",
        "Hobbies" : "Baseball"
}
>
```

6.5.8 Arrays

Objective: To create a collection by the name "food" and then insert documents into the "food" collection. Each document should have a "fruits" array.

Act:

```
db.food.insert({_id:1,fruits:[ 'banana','apple','cherry' ] })
db.food.insert({_id:2,fruits:[ 'orange','butterfruit','mango' ]})
db.food.insert({_id:3,fruits:[ 'pineapple','strawberry','grapes']});
db.food.insert({_id:4,fruits:[ 'banana','strawberry','grapes']});
db.food.insert({_id:5,fruits:[ 'orange','grapes']});
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.insert({_id:1,fruits:[ 'banana','apple','cherry' ] })
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:2,fruits:[ 'orange','butterfruit','mango' ]})
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:3,fruits:[ 'pineapple','strawberry','grapes']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:4,fruits:[ 'banana','strawberry','grapes']});
WriteResult({ "nInserted" : 1 })
> db.food.insert({_id:5,fruits:[ 'orange','grapes']});
WriteResult({ "nInserted" : 1 })
>
```

Outcome: Let us check if these documents are now in the "food" collection.

```
db.food.find({})
```

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({})
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 2, "fruits" : [ "orange", "butterfruit", "mango" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection which has the "fruits array" constituted of "banana", "apple" and "cherry".

Act:

```
db.food.find({fruits:['banana','apple','cherry']}).pretty()
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({fruits:['banana','apple','cherry']}).pretty()
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
>
```

Objective: To find those documents from the "food" collection which has the "fruits" array having "banana", as an element.

Act:

```
db.food.find({fruits:'banana'})
```

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({fruits:'banana'})
{ "_id" : 1, "fruits" : [ "banana", "apple", "cherry" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection which have the "fruits" array having "grapes" in the first index position. The index position begins at 0.

Act:

db.food.find({'fruits.1':'grapes'})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({'fruits.1':'grapes'})
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection where "grapes" is present in the 2nd index position of the "fruits" array.

Act:

db.food.find({'fruits.2':'grapes'})

Outcome:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({'fruits.2':'grapes'})
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "strawberry", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection where the size of the array is two. The size implies that the array holds only 2 values.

Act:

db.food.find({"fruits":{$size:2}})

Outcome:

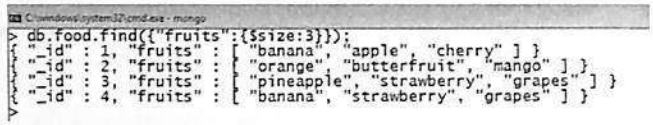
```
C:\windows\system32\cmd.exe - mongo
> db.food.find({"fruits":{$size:2}})
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
>
```

Objective: To find those documents from the "food" collection where the size of the array is three. The size implies that the array holds only 3 values.

Act:

db.food.find({"fruits":{$size:3}})

Outcome:

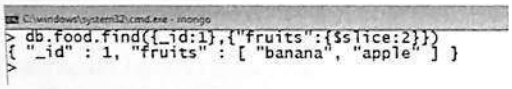


Objective: To find the document with (_id: 1) from the "food" collection and display the first two elements from the array "fruits".

Act:

db.food.find({_id:1},{"fruits":{$slice:2}})

Outcome:

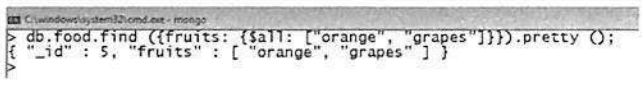


Objective: To find all documents from the "food" collection which have elements "orange" and "grapes" in the array "fruits".

Act:

db.food.find ({fruits: {$all: ["orange", "grapes"]}}).pretty ();

Outcome:

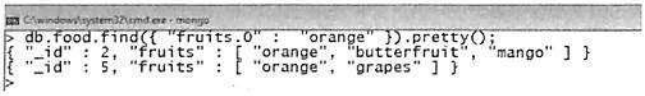


Objective: To find those documents from the "food" collection which have the element "orange" in the 0th index position in the array "fruits".

Act:

db.food.find({ "fruits.0" : "orange" }).pretty();

Outcome:

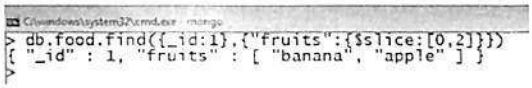


Objective: To find the document with (_id: 1) from the "food" collection and display two elements from the array "fruits", starting with the element at 0^{th} index position.

Act:

```
db.food.find({_id:1},{"fruits":{$slice:[0,2]}})
```

Outcome:

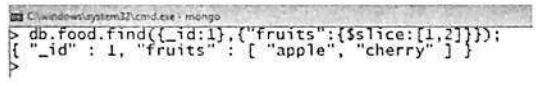


Objective: To find the document with (_id: 1) from the "food" collection and display two elements from the array "fruits", starting with the element at 1^{st} index position.

Act:

```
db.food.find({_id:1},{"fruits":{$slice:[1,2]}})
```

Outcome:

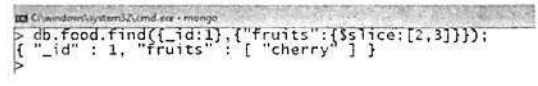


Objective: To find the document with (_id: 1) from the "food" collection and display three elements from the array "fruits", starting with the element at 2nd index position. Since we have only 3 elements in the array "fruits" for the document with _id:1, it displays only one element, the element at 2nd index position, that is, "cherry".

Act:

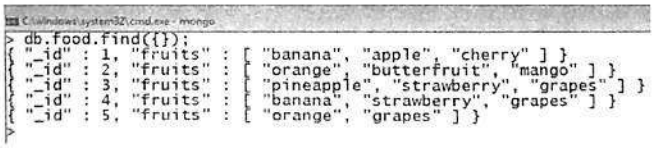
```
db.food.find({_id:1},{"fruits":{$slice:[2,3]}})
```

Outcome:



6.5.8.1 *Update on the Array*

Before we begin the update operations on the "fruits" array of the documents of "food" collection, let us take a look at the documents that we have in the "food" collection:



Objective: To update the document with "_id:4" and replace the element present in the 1st index position of the "fruits" array with "apple".

Act:

```
db.food.update({_id:4},{$set:{'fruits.1': 'apple'}})
```

```
> db.food.update({_id:4},{$set:{'fruits.1': 'apple'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: Let us take a look at how this update has changed our document.

```
> db.food.find({_id:4});
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
>
```

Objective: To update the document with "_id:1" and replace the element "apple" of the "fruits" array with "An apple".

Act:

```
db.food.update({_id:1, 'fruits':'apple'},{$set:{'fruits.$': 'An apple' }})
```

```
> db.food.update({_id:1, 'fruits':'apple'},{$set:{'fruits.$': 'An apple' }})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The document after update is as follows.

```
> db.food.find({_id:1});
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
>
```

Objective: To update the document with "_id:2" and push new key value pairs in the "fruits" array.

Act:

```
db.food.update({_id:2},{$push:{price:{orange:60,butterfruit:200,mango:120}}})
```

```
> db.food.update({_id:2},{$push:{price:{orange:60,butterfruit:200,mango:120}}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome:

```
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

6.5.8.2 Further Updates to the Array "fruits" ...

Before we do the updates to the documents in the food collection, let us look at the current state:

```
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:4" by adding an element "orange" to the list of elements in the array "fruits".

Act:

db.food.update({_id:4},{$addToSet:{fruits:"orange"}});

```
> db.food.update({_id:4},{$addToSet:{fruits:"orange"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The result after the execution of the statement is as follows.

```
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes", "orange" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:4" by popping an element from the list of elements present in the array "fruits". The element popped is the one from the end of the array.

Act:

db.food.update({_id:4},{$pop:{fruits:1}});

```
> db.food.update({_id:4},{$pop:{fruits:1}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The "food" collection after the execution of the statement is as follows.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "banana", "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:4" by popping an element from the list of elements present in the array "fruits". The element popped is the one from the beginning of the array.

Act:

db.food.update({_id:4},{$pop:{fruits:-1}});

```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4},{$pop:{fruits:-1}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The "food" collection after the execution of the above update statement is as follows.

```
C:\windows\system32\cmd.exe - mongo
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To update the document with "_id:3" by popping two elements from the list of elements present in the array "fruits". The elements popped are "pineapple" and "grapes".

The document with "_id:3" before the update is

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:3});
{ "_id" : 3, "fruits" : [ "pineapple", "strawberry", "grapes" ] }
>
```

Act:

```
db.food.update({_id:3},{$pullAll:{fruits: [ 'pineapple','grapes' ]}});
```

```
> db.food.update({_id:3},{$pullAll:{fruits: [ 'pineapple','grapes' ]}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The document with "_id:3" after the update is as follows:

```
> db.food.find({_id:4});
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
>
```

Objective: To update the documents having "banana" as an element in the array "fruits" and pop out the element "banana" from those documents.

The "food" collection before the update is as follows:

```
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "banana", "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "strawberry" ] }
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Act:

```
db.food.update({fruits:'banana'}, {$pull:{fruits:'banana'}})
```

```
> db.food.update({fruits:'banana'}, {$pull:{fruits:'banana'}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: The "food" collection after the update is as follows:

```
> db.food.find().pretty();
{ "_id" : 1, "fruits" : [ "An apple", "cherry" ] }
{ "_id" : 3, "fruits" : [ "strawberry" ] }
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
{ "_id" : 5, "fruits" : [ "orange", "grapes" ] }
{
        "_id" : 2,
        "fruits" : [
                "orange",
                "butterfruit",
                "mango"
        ],
        "price" : [
                {
                        "orange" : 60,
                        "butterfruit" : 200,
                        "mango" : 120
                }
        ]
}
>
```

Objective: To pull out an array element based on index position.

There is no direct way of pulling the array elements by looking up their index numbers. However a workaround is available. The document with "_id:4" in the food collection prior to the update is as follows:

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:4}).pretty();
{ "_id" : 4, "fruits" : [ "apple", "grapes" ] }
>
```

Act: The update statement is

db.food.update({_id:4}, {$unset : {"fruits.1" : null }}) ;
db.food.update({_id:4}, {$pull : {"fruits" : null}});

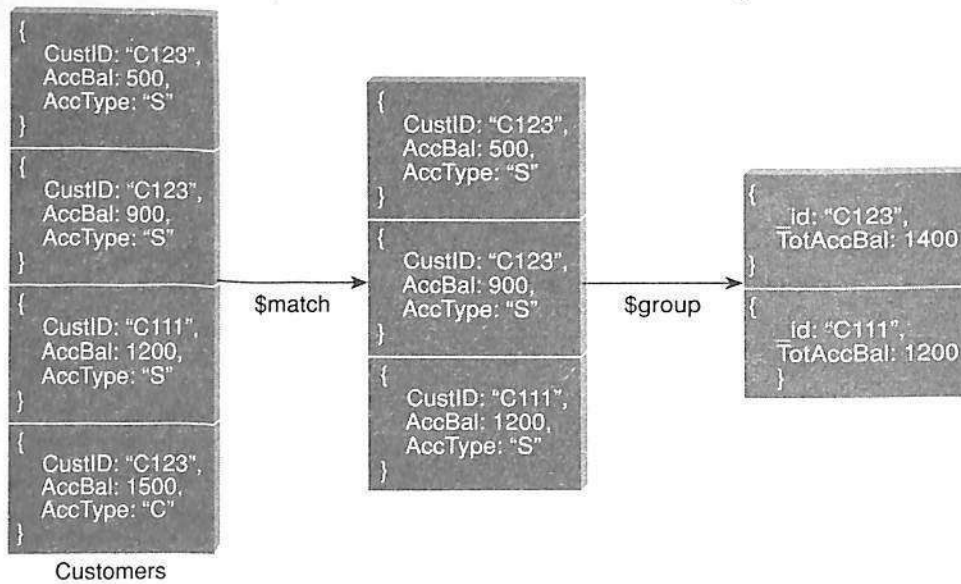
```
C:\windows\system32\cmd.exe - mongo
> db.food.update({_id:4}, {$unset : {"fruits.1" : null }}) ;
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.food.update({_id:4}, {$pull : {"fruits" : null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Outcome: After update, the document with _id:4 in the food collection is

```
C:\windows\system32\cmd.exe - mongo
> db.food.find({_id:4}).pretty();
{ "_id" : 4, "fruits" : [ "apple" ] }
>
```

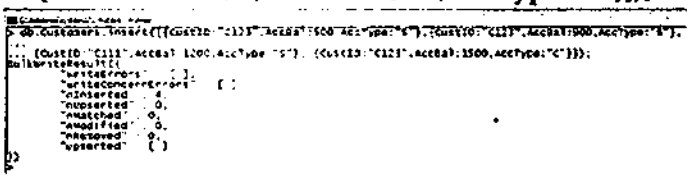
6.5.9 Aggregate Function

Objective: Consider the collection "Customers" as given below. It has four documents. We would like to filter out those documents where the "AccType" has a value other than "S". After the filter, we should be left with three documents where the "Acctype": "S". It is then required to group the documents on the basis of CustID and sum up the "AccBal" for each unique "CustID". This is similar to the output received with group by clause in RDBMS. Once the groups have been formed [as per the example below, there will be only two groups: (a) "CustID" : "C123" and (b) "CustID" : "C111"], filter and display that group where the "TotAccBal" column has a value greater than 1200.



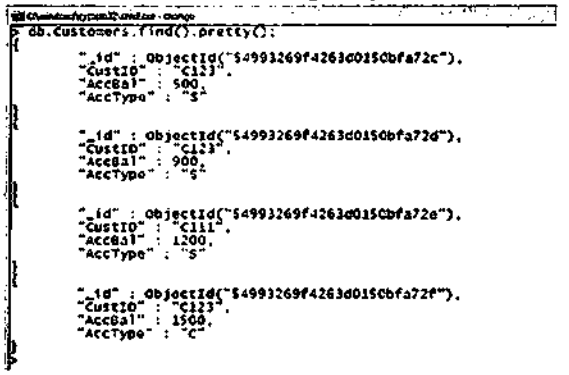
Let us start off by creating the collection "Customers" with the above displayed four documents:

```
db.Customers.insert([{CustID:"C123",AccBal:500,AccType:"S"},
{CustID:"C123", AccBal: 900, AccType:"S"},
{CustID:"C111", AccBal: 1200, AccType:"S"},
{CustID:"C123", AccBal: 1500, AccType:"C"}]);
```



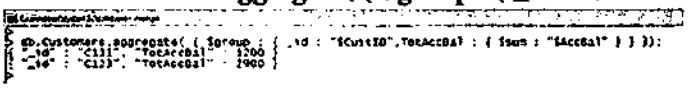
To confirm the presence of four documents in the "Customers" collection, use the below syntax:

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



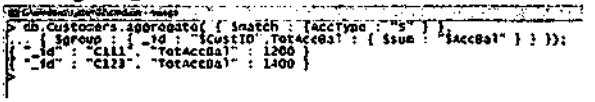
To group on "CustID" and compute the sum of "AccBal", use the below syntax:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } });
```



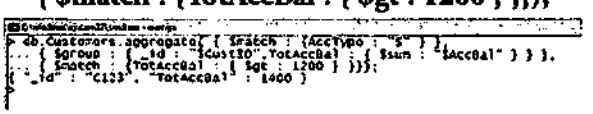
In order to first filter on "AccType:S" and then group it on "CustID" and then compute the sum of "AccBal", use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },
{ $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } });
```



In order to first filter on "AccType:S" and then group it on "CustID" and then to compute the sum of "AccBal" and then filter those documents wherein the "TotAccBal" is greater than 1200, use the below syntax:

```
db.Customers.aggregate( { $match : {AccType : "S" } },
{ $group : { _id : "$CustID",TotAccBal : { $sum : "$AccBal" } } },
{ $match : {TotAccBal : { $gt : 1200 } }});
```



To group on "CustID" and compute the average of the "AccBal" for each group:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $avg : "$AccBal" } } });
```

```
> db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $avg : "$AccBal" } } });
{ "_id" : "C111", "TotAccBal" : 1200 }
{ "_id" : "C123", "TotAccBal" : 966.6666666666666 }
>
```

To group on "CustID" and determine the maximum "AccBal" for each group:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $max : "$AccBal" } } });
```

```
>
> db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $max : "$AccBal" } } });
{ "_id" : "C111", "TotAccBal" : 1200 }
{ "_id" : "C123", "TotAccBal" : 1500 }
>
```

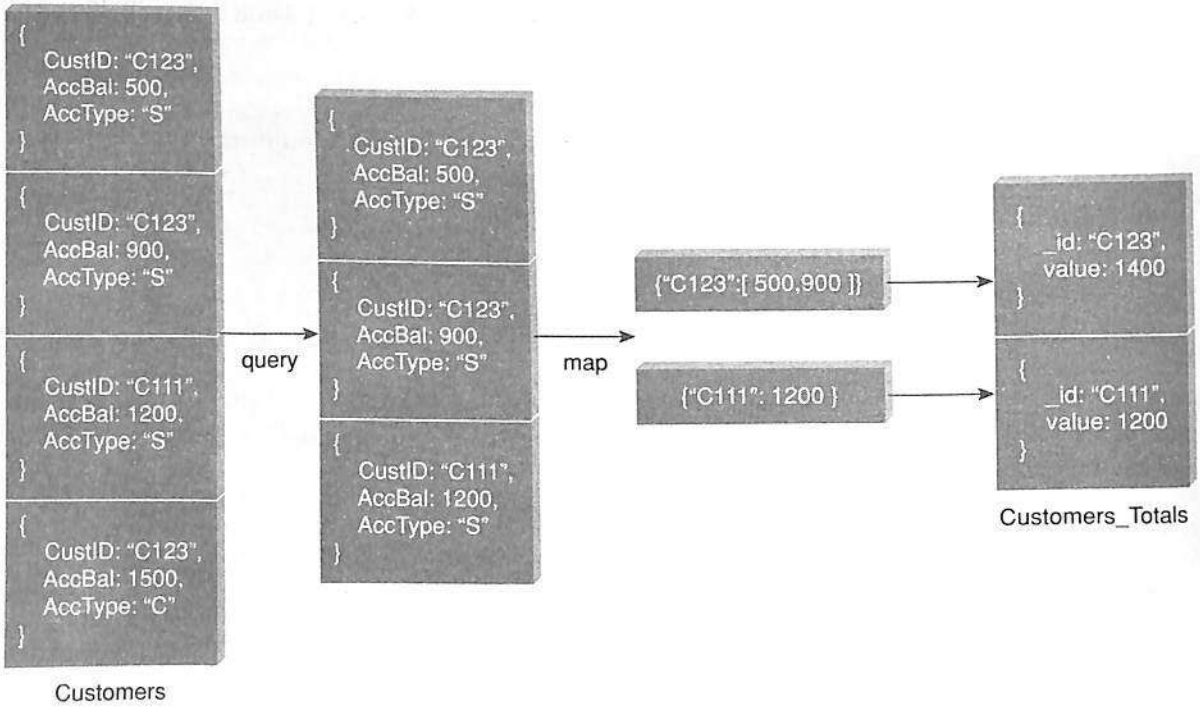
To group on "CustID" and determine the minimum "AccBal" for each group:

```
db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $min : "$AccBal" } } });
```

```
> db.Customers.aggregate( { $group : { _id : "$CustID",TotAccBal : { $min : "$AccBal" } } });
{ "_id" : "C111", "TotAccBal" : 1200 }
{ "_id" : "C123", "TotAccBal" : 500 }
>
```

6.5.10 MapReduce Function

Objective: Consider the collection "Customers" below. There are four documents. Run a query to filter out those documents where the key "AccType" has a value other than "S". Then for each unique CustID, prepare a list of AccBal values. For example, for CustID: "C123", the AccBals are 500,900. This task will be assigned to the mapper function. The output from the mapper function serves as the input to the reducer function. The reducer function then aggregates the AccBal for each CustID. For example, for CustID: "C123", the value is 1400, etc.



Given below is the syntax that we will use to accomplish the objective.

```
db.Customers.mapReduce (
map          →      function() { emit ( this.CustID, this.AccBal ); },
reduce       →      function(key, values) { return Array.sum (values ) },
                    {
query        →          query: { AccType: "S"},
output       →          out: "Customer_Totals"
                    }
                    )
```

Map Function

```
var map = function(){
emit ( this.CustID, this.AccBal );}
```

```
> var map = function(){
... emit ( this.CustID, this.AccBal );}
>
```

Reduce Function

```
var reduce = function(key, values){ return Array.sum(values) ; }
```

```
> var reduce = function(key, values){ return Array.sum(values) ; }
>
```

To execute the query

```
db.Customers.mapReduce(map, reduce,{out: "Customer_Totals", query:{AccType:"S"}});
```

```
> db.Customers.mapReduce(map, reduce,{out: "Customer_Totals", query:{AccType:"S"}});
{
        "result" : "Customer_Totals",
        "timeMillis" : 7,
        "counts" : {
                "input" : 3,
                "emit" : 3,
                "reduce" : 1,
                "output" : 2
        },
        "ok" : 1,
}
>
```

The output as archived in "Customer_Totals" collection:

```
> db.Customer_Totals.find().pretty();
{ "_id" : "C111", "value" : 1200 }
{ "_id" : "C123", "value" : 1400 }
>
```

6.5.11 Java Script Programming

Objective: To compute the factorial of a given positive number. The user is required to create a function by the name "factorial" and insert it into the "system.js" collection.

Before we proceed, a quick check on what is contained in the "system.js" collection:

```
> db.system.js.find();
>
```

As per the screenshot above, currently there are no functions in the system.js collection.

Act:

```
db.system.js.insert({_id:"factorial",
```

```
value:function(n)
{
      if (n==1)
      return 1;
      else
      return n * factorial(n-1);
}
      }
);
```

```
> db.system.js.insert({_id:"factorial",
... value:function(n)
... {
...  if (n==1)
... return 1;
...  else
...  return n * factorial(n-1);
... }
... });
WriteResult({ "nInserted" : 1 })
>
```

Confirm the presence of the "factorial" function in the system.js collection.

```
> db.system.js.find();
{ "_id" : "factorial", "value" : function (n)
{
if (n==1)
return 1;
else
return n * factorial(n-1);
} }
>
```

To execute the function "factorial", use the eval() method.

```
db.eval("factorial(3)");
```

```
> db.eval("factorial(3)");
6
>
```

```
db.eval("factorial(5)");
```

```
> db.eval("factorial(5)");
120
>
```

```
db.eval("factorial(1)");
```

```
> db.eval("factorial(1)");
1
>
```

6.5.12 Cursors in MongoDB

Objective: To create a collection by the name "alphabets" and insert documents in it containing two fields, "_id" and "alphabet". The values stored in the "alphabet" field should be "a", "b", "c", "d", etc. with one value stored per document. There should be 26 documents in all. We need to use cursor to iterate through the "alphabets" collection.

Note: "Alphabets" is the name of the collection and "alphabet" is the name of the field.

Act: To create the collection "alphabets" with its 26 documents.

```
db.alphabets.insert({_id:1,alphabet:"a"});
db.alphabets.insert({_id:2,alphabet:"b"});
```

```
db.alphabets.insert({_id:3,alphabet:"c"});
db.alphabets.insert({_id:4,alphabet:"d"});
db.alphabets.insert({_id:5,alphabet:"e"});
db.alphabets.insert({_id:6,alphabet:"f"});
db.alphabets.insert({_id:7,alphabet:"g"});
db.alphabets.insert({_id:8,alphabet:"h"});
db.alphabets.insert({_id:9,alphabet:"i"});
db.alphabets.insert({_id:10,alphabet:"j"});
db.alphabets.insert({_id:11,alphabet:"k"});
db.alphabets.insert({_id:12,alphabet:"l"});
db.alphabets.insert({_id:13,alphabet:"m"});
db.alphabets.insert({_id:14,alphabet:"n"});
db.alphabets.insert({_id:15,alphabet:"o"});
db.alphabets.insert({_id:16,alphabet:"p"});
db.alphabets.insert({_id:17,alphabet:"q"});
db.alphabets.insert({_id:18,alphabet:"r"});
db.alphabets.insert({_id:19,alphabet:"s"});
db.alphabets.insert({_id:20,alphabet:"t"});
db.alphabets.insert({_id:21,alphabet:"u"});
db.alphabets.insert({_id:22,alphabet:"v"});
db.alphabets.insert({_id:23,alphabet:"w"});
db.alphabets.insert({_id:24,alphabet:"x"});
db.alphabets.insert({_id:25,alphabet:"y"});
db.alphabets.insert({_id:26,alphabet:"z"});
```

```
C:\Windows\system32\cmd.exe - mongo
> db.alphabets.insert({_id:1,alphabet:"a"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:2,alphabet:"b"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:3,alphabet:"c"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:4,alphabet:"d"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:5,alphabet:"e"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:6,alphabet:"f"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:7,alphabet:"g"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:8,alphabet:"h"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:9,alphabet:"i"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:10,alphabet:"j"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:11,alphabet:"k"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:12,alphabet:"l"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:13,alphabet:"m"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:14,alphabet:"n"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:15,alphabet:"o"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:16,alphabet:"p"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:17,alphabet:"q"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:18,alphabet:"r"});
WriteResult({ "nInserted" : 1 })
```

```
> db.alphabets.insert({_id:19,alphabet:"s"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:21,alphabet:"u"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:22,alphabet:"v"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:23,alphabet:"w"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:24,alphabet:"x"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:25,alphabet:"y"});
WriteResult({ "nInserted" : 1 })
> db.alphabets.insert({_id:26,alphabet:"z"});
WriteResult({ "nInserted" : 1 })
>
```

Confirm the presence of 26 documents in the "alphabets" collection.

```
C:\windows\system32\cmd.exe - mongo
> db.alphabets.find();
{ "_id" : 1, "alphabet" : "a" }
{ "_id" : 2, "alphabet" : "b" }
{ "_id" : 3, "alphabet" : "c" }
{ "_id" : 4, "alphabet" : "d" }
{ "_id" : 5, "alphabet" : "e" }
{ "_id" : 6, "alphabet" : "f" }
{ "_id" : 7, "alphabet" : "g" }
{ "_id" : 8, "alphabet" : "h" }
{ "_id" : 9, "alphabet" : "i" }
{ "_id" : 10, "alphabet" : "j" }
{ "_id" : 11, "alphabet" : "k" }
{ "_id" : 12, "alphabet" : "l" }
{ "_id" : 13, "alphabet" : "m" }
{ "_id" : 14, "alphabet" : "n" }
{ "_id" : 15, "alphabet" : "o" }
{ "_id" : 16, "alphabet" : "p" }
{ "_id" : 17, "alphabet" : "q" }
{ "_id" : 18, "alphabet" : "r" }
{ "_id" : 19, "alphabet" : "s" }
{ "_id" : 20, "alphabet" : "t" }
Type "it" for more
>
```

A quick word on how the db.collection.find() method works. This is the primary method for read operation. In other words, it allows one to fetch the documents from the collection. To be able to access the documents, one needs to iterate the cursor.

However, in the mongo shell, if the returned cursor is not assigned to a variable using the var keyword, then cursor is automatically iterated up to 20 times to print the first 20 documents in the result.

```
C:\windows\system32\cmd.exe - mongo
> var myCursor = db.alphabets.find();
> myCursor;
{ "_id" : 1, "alphabet" : "a" }
{ "_id" : 2, "alphabet" : "b" }
{ "_id" : 3, "alphabet" : "c" }
{ "_id" : 4, "alphabet" : "d" }
{ "_id" : 5, "alphabet" : "e" }
{ "_id" : 6, "alphabet" : "f" }
{ "_id" : 7, "alphabet" : "g" }
{ "_id" : 8, "alphabet" : "h" }
{ "_id" : 9, "alphabet" : "i" }
{ "_id" : 10, "alphabet" : "j" }
{ "_id" : 11, "alphabet" : "k" }
{ "_id" : 12, "alphabet" : "l" }
{ "_id" : 13, "alphabet" : "m" }
{ "_id" : 14, "alphabet" : "n" }
{ "_id" : 15, "alphabet" : "o" }
{ "_id" : 16, "alphabet" : "p" }
{ "_id" : 17, "alphabet" : "q" }
{ "_id" : 18, "alphabet" : "r" }
{ "_id" : 19, "alphabet" : "s" }
{ "_id" : 20, "alphabet" : "t" }
Type "it" for more
>
```

Let us now look at designing manual cursors to iterate through the documents in the "alphabets" collection. We will use two methods with manual cursors: hasNext() and next(). We now quickly explain the two methods.

Method 1: hasNext() method. Return value: Boolean. The hasNext() method returns true if the cursor returned by the db.Collection.find() query can iterate further to return more documents.

Method 2: next() method. The next() method returns the next document in the cursor as returned by the db.collection.find() method.

```
C:\windows\system32\cmd.exe - mongo
> var myCur=db.alphabets.find({});
> while(myCur.hasNext()){
... var myRec=myCur.next();
... print("The alphabet is : " + myRec.alphabet);
... }
The alphabet is : a
The alphabet is : b
The alphabet is : c
The alphabet is : d
The alphabet is : e
The alphabet is : f
The alphabet is : g
The alphabet is : h
The alphabet is : i
The alphabet is : j
The alphabet is : k
The alphabet is : l
The alphabet is : m
The alphabet is : n
The alphabet is : o
The alphabet is : p
The alphabet is : q
The alphabet is : r
The alphabet is : s
The alphabet is : t
The alphabet is : u
The alphabet is : v
The alphabet is : w
The alphabet is : x
The alphabet is : y
The alphabet is : z
>
```

The same result can be obtained by iterating through the cursor using a forEach loop.

```
C:\windows\system32\cmd.exe - mongo
> var cur=db.alphabets.find({});
> var myRec;
> cur.forEach( function(myRec) {
... print("The alphabet is : " + myRec.alphabet);
...
... });
The alphabet is : a
The alphabet is : b
The alphabet is : c
The alphabet is : d
The alphabet is : e
The alphabet is : f
The alphabet is : g
The alphabet is : h
The alphabet is : i
The alphabet is : j
The alphabet is : k
The alphabet is : l
The alphabet is : m
The alphabet is : n
The alphabet is : o
The alphabet is : p
The alphabet is : q
The alphabet is : r
The alphabet is : s
The alphabet is : t
The alphabet is : u
The alphabet is : v
The alphabet is : w
The alphabet is : x
The alphabet is : y
The alphabet is : z
>
```

6.5.13 Indexes

Assume the collection with the following documents:

```
> db.books.find().pretty();
{
        "_id" : 6,
        "Category" : "Machine Learning",
        "Bookname" : "Machine Learning for Hackers",
        "Author" : "Drew Conway",
        "qty" : 25,
        "price" : 400,
        "rol" : 30,
        "pages" : 350
}
{
        "_id" : 7,
        "Category" : "Web Mining",
        "Bookname" : "Mining the Social Web",
        "Author" : "Matthew A.Russell",
        "qty" : 55,
        "price" : 500,
        "rol" : 30,
        "pages" : 250
}
{
        "_id" : 8,
        "Category" : "Python",
        "Bookname" : "Python for Data Analysis",
        "Author" : "Wes McKinney",
        "qty" : 8,
        "price" : 150,
        "rol" : 20,
        "pages" : 150
}
{
        "_id" : 9,
        "Category" : "Visualization",
        "Bookname" : "Visualizing Data",
        "Author" : "Ben Fry",
        "qty" : 12,
        "price" : 325,
        "rol" : 6,
        "pages" : 450
}
{
        "_id" : 10,
        "Category" : "Web Mining",
        "Bookname" : "Algorithms for the intelligent web",
        "Author" : "Haralambos Marmanis",
        "qty" : 5,
        "price" : 850,
        "rol" : 10,
        "pages" : 120
}
```

Create an index on the key "Category" in the "books" collection.

```
> db.books.ensureIndex({"Category":1});
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Check on the status, that is, number and name of the indexes:

```
> db.books.stats();
{
        "ns" : "test.books",
        "count" : 5,
        "size" : 1200,
        "avgObjSize" : 240,
        "storageSize" : 8192,
        "numExtents" : 1,
        "nindexes" : 2,
        "lastExtentSize" : 8192,
        "paddingFactor" : 1,
        "systemFlags" : 1,
```

```
        "userFlags" : 1,
        "totalIndexSize" : 16352,
        "indexSizes" : {
                "_id_" : 8176,
                "Category_1" : 8176
        },
        "ok" : 1
}
```

Get the list of all indexes on the "books" collection:

```
> db.books.getIndexes();
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "test.books"
        },
        {
                "v" : 1,
                "key" : {
                        "Category" : 1
                },
                "name" : "Category_1",
                "ns" : "test.books"
        }
]
```

To use the index on "Category" in the "books" collection, use the hint method:

```
> db.books.find({"Category":"Web Mining"}).pretty().hint({"Category":1});
{
        "_id" : 7,
        "Category" : "Web Mining",
        "Bookname" : "Mining the Social Web",
        "Author" : "Matthew A.Russell",
        "qty" : 55,
        "price" : 500,
        "rol" : 30,
        "pages" : 250
}
{
        "_id" : 10,
        "Category" : "Web Mining",
        "Bookname" : "Algorithms for the intelligent web",
        "Author" : "Haralambos Marmanis",
        "qty" : 5,
        "price" : 850,
        "rol" : 10,
        "pages" : 120
}
```

Check the explain plan to get a deeper understanding on the use of index.

```
> db.books.find({"Category":"Web Mining"}).pretty().hint({"Category":1}).explain();
{
        "cursor" : "BtreeCursor Category_1",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 2,
        "nscanned" : 2,
        "nscannedObjectsAllPlans" : 2,
        "nscannedAllPlans" : 2,
        "scanAndOrder" : false,
        "indexOnly" : false,
        "nYields" : 0,
        "nChunkSkips" : 0,
```

```
        "millis" : 0,
        "indexBounds" : {
                "Category" : [
                        [
                                "Web Mining",
                                "Web Mining"
                        ]
                ]
        },
        "server" : "PUNITP123103L:27017",
        "filterSet" : false
}
```

Let us look at the case of covered index. Observe that the "indexOnly" property will be set to true for covered index.

```
> db.books.find({"Category":"Web Mining"},{"Category":1,_id:0}).pretty().hint({"Category":1}).explain();
{
        "cursor" : "BtreeCursor Category_1",
        "isMultiKey" : false,
        "n" : 2,
        "nscannedObjects" : 0,
        "nscanned" : 2,
        "nscannedObjectsAllPlans" : 0,
        "nscannedAllPlans" : 2,
        "scanAndOrder" : false,
        "indexOnly" : true,
        "nYields" : 0,
        "nChunkSkips" : 0,
        "millis" : 0,
        "indexBounds" : {
                "Category" : [
                        [
                                "Web Mining",
                                "Web Mining"
                        ]
                ]
        },
        "server" : "PUNITP123103L:27017",
        "filterSet" : false
}
```

In order to have the index cover the query, ensure that only those columns are projected on which the index is built. In the above example, the index is built on the "Category" column, and "Category" is the only column that is projected. Even the identifier (_id) is suppressed.

6.5.14 MongoImport

This command used at the command prompt imports CSV (Comma Separated Values) or TSV (Tab Separated Values) files or JSON (Java Script Object Notation) documents into MongoDB.

Objective: Given a CSV file "sample.txt" in the D: drive, import the file into the MongoDB collection, "SampleJSON". The collection is in the database "test".

The "sample.txt" file is as follows:

_id,FName,LName
1,Samuel,Jones
2,Virat,Kumar
3,Raul,"A Simpson"
4,,"Andrew Simon"

Act:

At the command prompt, execute the following command:

Mongoimport --db test --collection SampleJSON --type csv --headerline --file d:\sample.txt

On successful execution of the command, the message at the prompt will be as follows:

```
connected to: 127.0.0.1
2015-02-20T21:09:27.301+0530 imported 4 objects
```

Output: To confirm the output, log into MongoDB shell and navigate to the "SampleJSON" collection in the "test" database.

The following are the JSON documents in the collection:

```
> db
test
> show collections
Customers
SampleJSON
books
fs.chunks
fs.files
persons
system.indexes
usercounters
users
> db.SampleJSON.find().pretty();
{ "_id" : 1, "FName" : "Samuel", "LName" : "Jones" }
{ "_id" : 2, "FName" : "Virat", "LName" : "Kumar" }
{ "_id" : 3, "FName" : "Raul", "LName" : "A Simpson" }
{ "_id" : 4, "FName" : "", "LName" : "Andrew Simon" }
>
```

6.5.15 MongoExport

This command used at the command prompt exports MongoDB JSON documents into CSV (Comma Separated Values) or TSV (Tab Separated Values) files or JSON (Java Script Object Notation) documents.

Objective: This command used at the command prompt exports MongoDB JSON documents from "Customers" collection in the "test" database into a CSV file "Output.txt" in the D: drive.

Given below is a snapshot of the JSON documents in the "Customers" collection of the "test" database.

```
> db
test
> show collections
Customers
SampleJSON
books
fs.chunks
fs.files
persons
system.indexes
usercounters
users
> db.Customers.find().pretty();
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5b"),
        "CustID" : "C123",
        "AccBal" : 500,
        "AccType" : "S"
}
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5c"),
        "CustID" : "C123",
        "AccBal" : 900,
        "AccType" : "S"
}
```

```
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5d"),
        "CustID" : "C111",
        "AccBal" : 1200,
        "AccType" : "S"
}
{
        "_id" : ObjectId("54df6d4f46a31d28183b9a5e"),
        "CustID" : "C123",
        "AccBal" : 1500,
        "AccType" : "C"
}
>
```

Act: At the command prompt, execute the following command:

Mongoexport --db test --collection Customers --csv --fieldFile d:\fields.txt --out d:\output.txt

Before executing this command, ensure that you have created a "fields.txt" with a format defined as follows. The "fields.txt" file:

CustID
AccBal
AccType

For the MongoExport command to execute successfully, ensure that the fields are spelt as is in the MongoDB collection. The case also has to be maintained. It is mandatory to ensure that only one field name is placed per line.

On successful execution of the command, the message at the prompt will be as follows:

```
connected to: 127.0.0.1
exported 4 records
```

Output: To confirm the output, navigate to the D: drive and check the file "Output.txt".

"Output.txt"
CustID,AccBal,AccType
"C123",500.0,"S"
"C123",900.0,"S"
"C111",1200.0,"S"
"C123",1500.0,"C"

6.5.16 Automatic Generation of Unique Numbers for the "_id" Field

Step 1: Run the insert() method on a new collection "usercounters". This is to start off with an initial value of 0 for the "seq" field.

```
db.usercounters.insert(
{
_id: "empid",
seq:0
}
)
```

Step 2: Create a user-defined function "getnextseq". This method will invoke "findAndModify()" method on the "usercounters" collection. This is to increment the value of seq field by 1 and update the same in "usercounters" collection.

```
function getnextseq(name) {
var ret=db.usercounters.findAndModify(
{
query: {_id:name},
update: {$inc:{seq:1}},
new:true
}
);
return ret.seq;
}
```

Step 3: Run the insert() method on the collection where you need to have the "_id" field and get the uniquely generated number. Notice the call to getnextseq() method as value to _id. The return value from the getnextseq() method becomes the value of _id.

```
db.users.insert(
{
_id:getnextseq("empid"),
Name: "sarah jane"
}
)
```

REMIND ME

- MongoDB is a non-relational, open source, distributed database.
- It stores data into JSON (Java Script Object Notation) documents.
- It adheres to CP (Consistency and Partition Tolerant) traits of Brewer's CAP theorem.
- It has NO support for multi-statement transactions.
- It supports embedded documents.
- It practices automatic sharding.

POINT ME (BOOK)

- MongoDB: The definitive guide by Kristina Chodorow, Michael Dirolf, O'Reilly Media.

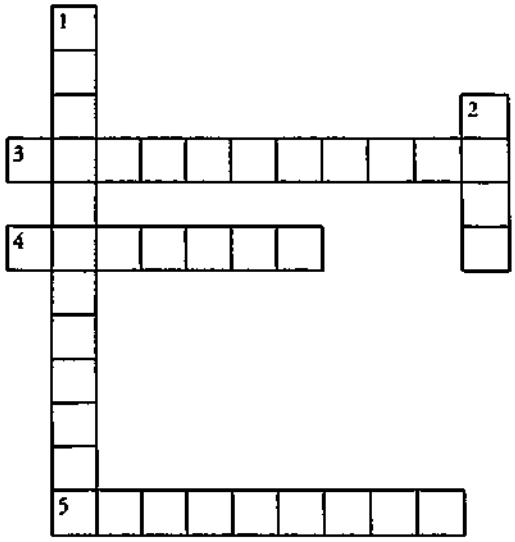
CONNECT ME (INTERNET RESOURCES)

- http://www.mongodb.org/
- https://university.mongodb.com/
- http://www.tutorialspoint.com/mongodb/

TEST ME

A. Crossword

Puzzle on MongoDB



Across

3. MongoDB database stores its data is _____.
4. MongoDB uses _____ schemes.
5. A collection holds one or more _____.

Down

1. MongoDB uses _____ files.
2. MongoDB uses _____, a binary object format similar to, but more expensive than JSON.

Answer:

Across

3. Collections
4. Dynamic
5. Documents

Down

1. Memorymapped
2. BSON

B. Pick the Right Choice

1. MongoDB supports dynamic schema design.
   (a) False (b) True
2. MongoDB supports query joins between collections.
   (a) False (b) True
3. Which of the following MongoDB conditional operator is not a valid operator?
   (a) $lt (c) $gt
   (b) $ltu (d) $lte
4. '$unset' is used with
   (a) Insert (c) Delete
   (b) Update

5. MongoDB is supported by
   (a) Perl
   (b) Python
   (c) PHP
   (d) All the above
6. MongoDB is _____.
   (a) RDBMS
   (b) Object-oriented DBMS
   (c) Document-oriented DBMS
   (d) Key–value store
7. 'MongoImport' command is used _____.
   (a) For multiple command insertion.
   (b) To import content from a JSON, CSV, or TSV export created by MongoExport.
   (c) For multiple command import.
8. Which of the following is the correct command to insert data into MongoDB? Assume that document is a valid JSON document.
   (a) db.Students.insert(document)
   (b) db.Students.insert().(document)
   (c) Students.insert(document)
9. MongoDB documents are represented as _____.
   (a) XML
   (b) JSON
   (c) DOCUMENT
10. MongoDB supports unique indexes just like most other relational databases.
    (a) True
    (b) False
11. Which of the following command creates an index, where mobile_no is a field in the collection, employees?
    (a) db.employees.SetIndex({ "mobile_no": 1 })
    (b) db.employees.ensureIndex({ "mobile_no": 1 })
    (c) employees.SetIndex({ "mobile_no": 1 })
12. Which of the following command is correct when you want to fetch documents from a collection for "only those employees whose salary is either 8500 or 10,000"?
    (a) db.employees.find.sort({"salary" :{$in :[8500,10000]}})
    (b) db.employees.find({"salary" :{"$in :[8500,10000]"}})
    (c) db.employees.find({"salary" :{$in :[8500,10000]}})
13. Which of the following is the command equivalent to
    Select first_name,salary,date_of_join from employees where designation="Manager";
    (a) db.employees.find({"designation":"Manager"},{"first_name" : 1;"salary":1;"date_of_join":1})
    (b) db.employees.find({"designation:Manager"},{"first_name" : 1,"salary":1,"date_of_join":1})
    (c) db.employees.find({"designation":"Manager"},{"first_name" : 1,"salary":1,"date_of_join":1})
14. MongoDB enforces attribute similarity across documents in a collection.
    (a) True
    (b) False
15. The maximum BSON document size is
    (a) 8 megabytes
    (b) 4 megabytes
    (c) 32 megabytes
    (d) 16 megabytes
16. Core MongoDB operations are
    (a) Create, Select, Update, Delete
    (b) Create, Read, Update, Delete
    (c) Create, Read, Update, Drop

17. Which of the following command provides you with a list of all the databases in MongoDB?
    (a) show databases
    (b) show dbs
    (c) show all dbs
    (d) None of the above
18. If we want to remove the document from the collection 'employees' which contains the 'first_name' as "John" then which of the following MongoDB command can be used:
    (a) db.userdetails.remove({})
    (b) db.employees.remove({ "first_name : John" })
    (c) db.employees.remove({ "first_name" : "John" })
19. What does the following command do?
    db.sample.find().limit(10)
    (a) Show 10 documents randomly from the collection sample
    (b) Show only first 10 documents from the collection sample
    (c) Repeats the first document 10 times
20. Which one of the following is equivalent to
    Select * from employees order by salary
    (a) db.employees.find().sort({"salary:1"})
    (b) db.employees.sort({"salary":1})
    (c) db.employees.find().sort({"salary":1})
21. Which command in MongoDB is equivalent to SQL select?
    (a) search()
    (b) find()
    (c) document()
22. Which of the following is equivalent to
    select first_name,salary from employees where designation="Manager";
    Assume that there are three columns first_name,salary,date_of_join.
    (a) db.employees.find({"designation:Manager"},{"date_of_join" : 0})
    (b) db.employees.find({"designation:Manager"},{"date_of_join" : 1})
    (c) db.employees.find({"designation":"Manager"},{"date_of_join" : 0})
23. Which of the following statement is equivalent to the SQL command – Select * from employees where designation = "Manager"?
    (a) employees.find({"designation":"manager"}
    (b) db.employees.find({"designation:manager"}
    (c) db.employees.find({"designation": "manager"}
24. Which of the following is the correct command to update a document?
    (a) db.books.update({ item: "book", qty: { $gt: 7 } }, { $set: { x: 5 }, $inc: { y: 8} })
    (b) db.books.find().update({ item: "book", qty: { $gt: 7 } }, { $set: { x: 5 }, $inc: { y: 8} })
    (c) db.books.update({ item: "book", qty: { $gt: 7 } }, { $set: { x: 5 }, $inc: { y: 8} } , { multi: true })
25. Which one of the following is equivalent to
    Select * from employees order by salary desc;
    (a) db.employees.find().sort({"salary":1})
    (b) db.employees.find().sort({"salary":-1})
    (c) db.employees.sort({"salary":-1})
26. Which of the following command is correct when you want to fetch documents from a collection for only those employees whose salary is either 7500 or date of joining is 17/10/2009?
    (a) db.employees.find({ "salary" : "7500" , ($or : [{ "date_of_join" : "17/10/2009" }] }))
    (b) db.employees.find({ "salary" : "7500" , $or : [{ "date_of_join" : "17/10/2009" }] })
    (c) db.employees.find().sort({ "salary" : "7500" , $or : [{ "date_of_join" : "17/10/2009" }] })

27. What does the following command do?
    db.employees.find().skip(5).limit(5)
    (a) Skips first five documents and then shows just next five documents
    (b) Shows just next five documents
    (c) Skips first five documents and then shows the sixth one five times
28. Which of the following command is correct when you want to fetch documents from collection demo, where value of a field 'interest' is null?
    (a) db.demo.find({ "interest" : null })
    (b) db.demo.find().sort({ "interest" : null })
    (c) db.demo.find({ "interest : null" })

Answers:

1. True
2. False
3. (b)
4. (b)
5. (d)
6. (c)
7. (b)
8. (a)
9. (b)
10. (a)
11. (b)
12. (c)
13. (c)
14. (b)
15. (d)
16. (b)
17. (b)
18. (c)
19. (b)
20. (c)
21. (b)
22. (c)
23. (c)
24. (a)
25. (b)
26. (c)
27. (a)
28. (a)

C. Unsolved Exercises

1. Enumerate few features of MongoDB.
2. List the difference between SQL and MongoDB.
3. Explain Map Reduce programming in MongoDB with a suitable example.
4. What is a cursor? How is a cursor implemented in MongoDB. Explain with a suitable example.
5. What is the significance of _id key in MongoDB?

ASSIGNMENTS FOR HANDS-ON PRACTICE

ASSIGNMENT 1

Objective: **To practice MapReduce programming in MongoDB.**

Step 1: Execute the below statements at the MongoDB shell prompt.

db.books.save({ _id:1,Category:"Machine Learning", Bookname:"Machine Learning for Hackers", Author:"Drew Conway",qty:25,price:400,rol:30,pages:350});

db.books.save({ _id:2,Category:"Business Intelligence", Bookname:"Fundamentals of Business Analytics", Author:"Seema Acharya",qty:55,price:500,rol:30,pages:250});

db.books.save({ _id:3,Category:"Analytics", Bookname:" Competing on Analytics", Author:"Thomas Davenport",qty:8,price:150,rol:20,pages:150});

db.books.save({ _id:4,Category:"Visualization", Bookname:"Visualizing Data", Author:"Ben Fry",qty:12, price:325,rol:6,pages:450});