



KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act 1956)
Coimbatore-641 021
(For the candidates admitted from 2018 onwards)
DEPARTMENT OF COMPUTER SCIENCE, CA & IT

SUBJECT CODE : 18CSP103
SEMESTER : I

SUBJECT : BIG DATA ANALYTICS
CLASS : I M.Sc. CS L T P = 4 0 0

SCOPE

The course provides grounding in basic and advanced methods to big data technology and tools, including MapReduce and Hadoop and its ecosystem.

OBJECTIVES

On successful completion of the course the student should

- able to apply Hadoop ecosystem components.
- able to participate data science and big data analytics projects

UNIT -I

What is big data – why big data – convergence of key trends – unstructured data – industry examples of big data – web analytics – big data and marketing – fraud and big data – risk and big data – credit risk management – big data and algorithmic trading – big data and healthcare – big data in medicine – advertising and big data – big data technologies - open source technologies – cloud and big data – mobile business intelligence – Crowd sourcing analytics – inter and trans firewall analytics

UNIT-II

History of Hadoop- The Hadoop Distributed File System – Components of Hadoop- Analyzing the Data with Hadoop- Scaling Out- Hadoop Streaming- Design of HDFS- How Map Reduce Works-Anatomy of a Map Reduce Job run-Failures-Job Scheduling- Shuffle and Sort – Task execution - Map Reduce Types and Formats- Map Reduce Features

UNIT-III

Hbase – data model and implementations – Hbase clients – Hbase examples – praxis. Cassandra – cassandra data model – cassandra examples – cassandra clients – Hadoop integration. Pig – Grunt – pig data model – Pig Latin – developing and testing Pig Latin scripts. Hive – data types and file formats – HiveQL data definition – HiveQL data manipulation – HiveQL queries.

UNIT-IV

Introduction to NoSQL – aggregate data models – aggregates – key-value and document data models – relationships– schemaless databases – materialized views – distribution models -peer-peer replication –consistency – relaxing consistency – version stamps – partitioning and combining – composing map-reduce calculations -Document based Database – MongoDB- Introduction- Data Model- Working with data- Replication &Sharding- Development

UNIT-V

Graph databases Neo4J- Key concept and characteristics-Modelling data for neo4j- Importing data into neo4j-Visualizations neo4j-Cypher Query Language-Data visualization- Creating Visual analytics with Tableau-Connecting your data-Creating Calculation-Using maps-Dashboard-Stories

SUGGESTED READINGS

1. Tom White, “Hadoop: The Definitive Guide”
2. The Definitive Guide to MongoDB
3. Rik Van Bruggen, “Learning Neo4j”
4. Daniel G.Murray, “Tableau Your Data!: Fast and Easy Visual Analysis with Tableau Software”
5. Dirk deRoos, Paul Zikopoulos, Bruce Brown, Roman B. Melnyk,RafaelCoss, “Hadoop For Dummies”
6. GauravVaish, “Getting Started with NoSQL”
7. Pramod J. Sadalage, Martin Fowler, “NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence”
8. Joshua N. Milligan, “Learning Tableau”

ESE MARKS ALLOCATION

1.	Section A 20 x 1 = 20	20
2.	Section B 5 x 6 = 30 Either 'A' or 'B' choice	30
3.	Section C 1 x 10 = 10 Compulsory Question	10
	Total	60

**KARPAGAM ACADEMY OF HIGHER EDUCATION**

(Coimbatore-641 021)

(For the candidates admitted from 2018 onwards)

DEPARTMENT OF COMPUTER SCIENCE, CA & IT**STAFF NAME: D.MANJULA****SUBJECT NAME: BIG DATA ANALYTICS****SUB.CODE: 18CSP103****SEMESTER: I****CLASS : I M.SC CS****LECTURE PLAN****DEPARTMENT OF COMPUTER SCIENCE**

S.No.	Lecture Duration (Period)	Topics to be Covered	Support Materials
Unit – I			
1.	1	What Is Big Data, Why Big Data	T1:1-6
2.	1	Convergence of key trends, unstructured data	T1:7-8, T1:11-14
3.	1	Industry examples of big data, web analytics	T1:19,23
4.	1	Big data and marketing, fraud and big data	T1:27-36
5.	1	Risk and big data, credit risk management	T1:37-39
6.	1	Big data and algorithmic trading, big data and healthcare	T1:40-49
7.	1	Big data in medicine, advertising and big data	T1:51-57
8.	1	Big data technologies, open source technologies	T1:61,67-68
9.	1	Cloud and big data, mobile business intelligence	T1:69,73-75
10.	1	Crowd sourcing analytics, inter and trans firewall analytics	T1:76-79
11.	1	Recapitulation and discussion of important questions	
Total No. of Hours Planned for Unit-I			11
Unit – II			
1.	1	History of Hadoop	T2:12
2.	1	The Hadoop Distributed File System, Components of Hadoop	T2:
3.	1	Analyzing the Data with Hadoop, Scaling Out	T2:22-37
4.	1	Hadoop Streaming, Design of HDFS	T2:33-44
5.	1	How Map Reduce Works, Anatomy of a Map Reduce Job run	T2: 185-192
6.	1	Failures, Job Scheduling, Shuffle and Sort	T2:193-202
7.	1	Task execution, Reduce Types and Formats, - Map Reduce	T2:203,209,247

		Features	
8.		Recapitulation and Discussion of important questions	
Total No. of Hours Planned for Unit-II			8
Unit – III			
1.	1	Hbase, data model and implementations	T2:575,578
2.	1	Hbase clients, Hbase examples, praxis	T2:584,600-601
3.	1	Cassandra, cassandra data model, cassandra examples	W6
4.	1	cassandra clients, Hadoop integration	W6
5.	1	Pig, Grunt, pig data model	T2:423,426,
6.	1	Pig Latin, developing and testing , Pig Latin scripts	T2:432,447
7.	1	Hive, data types and file formats	T2:486
8.	1	HiveQL data definition, HiveQL data manipulation, HiveQL queries	T2:503-509
9.		Recapitulation and Discussion of important questions	
Total No. of Hours Planned for Unit-III			9
Unit - IV			
1.	1	Introduction to NoSQL, aggregate data models, aggregates	T3:26-32
2.	1	key-value and document data models, relationships, schemaless databases	T3:32,36-40
3.	1	Materialized views, distribution models, peer-peer replication	T3:40-44,50-51
4.	1	Consistency, relaxing consistency, version stamps	T3:54,56-62,65-68
5.	1	partitioning and combining, composing map, reduce calculations	T3:71-79
6.	1	Document based Database, MongoDB-Introduction	T3:86-87.T4:11
7.	1	Data Model, Working with data	T4:16-20
8.	1	Replication &Sharding- Development	T4:169-185,231
9.	1	Recapitulation and Discussion of important questions	
Total No. of Hours Planned for Unit-IV			9
Unit - V			
1.	1	Graph databases Neo4J, Key concept and characteristics	W1
2.	1	Modelling data for neo4j, Importing data into neo4j	W1
3.	1	Visualizations neo4j, Cypher Query Language	W2
4.	1	Data visualization	W2
5.	1	Creating Visual analytics with Tableau	W3
6.	1	Connecting your data, Creating Calculation	W4
7.	1	Using maps, Dashboard, Stories	W5
8.	1	Recapitulation and Discussion of important questions	

9.	1	Discussion of previous ESE question paper	
10.	1	Discussion of previous ESE question paper	
11.	1	Discussion of previous ESE question paper	
		Total No. of Hours Planned for Unit-V	11
		Total No. of periods	48

TEXTBOOKS:

T1:Big Data, Big Analytics, Michael Minelli, Michele Chambers Ambiga Dhiraj, Published by John Wiley & Sons, Inc., Hoboken, New Jersey,2013.

T2: Hadoop-The.Definitive.Guide,4.edition,a Tom White, April-2015

T3:Pramod J. Sadalage, Martin Fowler, “NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence,2013

T4:The Definitive Guide to Mongoddb, Kristina Chodorow,2013, O’Reilly Media

WEBSITES:

W1:<https://neo4j.com/docs/developer-manual/current/introduction/graphdb-concepts/>

W2: <https://neo4j.com/developer/guide-data-visualization/>

W3: <https://www.coursera.org/learn/dataviz-visual-analytics>

W4: https://www.tutorialspoint.com/tableau/tableau_numeric_calculations.htm

W5: <https://www.tableau.com/solutions/maps>

W6: <https://www.tutorialspoint.com/cassandra/>

UNIT I

Syllabus

What is big data – why big data – convergence of key trends – unstructured data – industry examples of big data – web analytics – big data and marketing – fraud and big data – risk and big data – credit risk management – big data and algorithmic trading – big data and healthcare – big data in medicine – advertising and big data – big data technologies - open source technologies – cloud and big data – mobile business intelligence – Crowd sourcing analytics – inter and trans firewall analytics

What is big data?

Big data means really a big data, it is a collection of large datasets that cannot be processed using traditional computing techniques. Big data is not merely a data, rather it has become a complete subject, which involves various tools, techniques and frameworks.

What Comes Under Big Data?

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

- **Black Box Data** : It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
- **Social Media Data** : Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
- **Stock Exchange Data** : The stock exchange data holds information about the 'buy' and 'sell' decisions made on a share of different companies made by the customers.
- **Power Grid Data** : The power grid data holds information consumed by a particular node with respect to a base station.
- **Transport Data** : Transport data includes model, capacity, distance and availability of a vehicle.

- **Search Engine Data** : Search engines retrieve lots of data from different databases.

Thus Big Data includes huge volume, high velocity, and extensible variety of data. The data in it will be of three types.

- **Structured data** : Relational data.
- **Semi Structured data** : XML data.
- **Unstructured data** : Word, PDF, Text, Media Logs.

Big data is often boiled down to a few varieties including social data, machine data, and transactional data. Social media data is providing remarkable insights to companies on consumer behavior and sentiment that can be integrated with CRM data for analysis, with 230 million tweets posted on Twitter per day, 2.7 billion Likes and comments added to Facebook every day, and 60 hours of video uploaded to YouTube every minute (this is what we mean by velocity of data). Machine data consists of information generated from industrial equipment, real-time data from sensors that track parts and monitor machinery (often also called the Internet of Things), and even web logs that track user behavior online. At arcplan client CERN, the largest particle physics research center in the world, the Large Hadron Collider (LHC) generates 40 terabytes of data every second during experiments. Regarding transactional data, large retailers and even B2B companies can generate multitudes of data on a regular basis considering that their transactions consist of one or many items, product IDs, prices, payment information, manufacturer and distributor data, and much more. Major retailers like Amazon.com, which posted \$10B in sales in Q3 2011, and restaurants like US pizza chain Domino's, which serves over 1 million customers per day, are generating petabytes of transactional big data. The thing to note is that big data can resemble traditional structured data or unstructured, high frequency information.

Big Data Analytics

Big (and small) Data analytics is the process of examining data—typically of a variety of sources, types, volumes and / or complexities—to uncover hidden patterns, unknown correlations, and other useful information. The intent is to find business insights that were not previously possible or were missed, so that better decisions can be made. Big Data analytics uses a wide variety of advanced analytics to provide

1. Deeper insights. Rather than looking at segments, classifications, regions, groups, or other summary levels you'll have insights into all the individuals, all the products, all the parts, all the events, all the transactions, etc.
2. Broader insights. The world is complex. Operating a business in a global, connected economy is very complex given constantly evolving and changing conditions. As humans, we simplify conditions so we can process events and

understand what is happening. But our best-laid plans often go astray because of the estimating or approximating. Big Data analytics takes into account all the data, including new data sources, to understand the complex, evolving, and interrelated conditions to produce more accurate insights.

3. Frictionless actions. Increased reliability and accuracy that will allow the deeper and broader insights to be automated into systematic actions.
Advanced Big data analytics Big data analytic applications

Why Big data?

1. Understanding and Targeting Customers

This is one of the biggest and most publicized areas of big data use today. Here, big data is used to better understand customers and their behaviors and preferences. Companies are keen to expand their traditional data sets with social media data, browser logs as well as text analytics and sensor data to get a more complete picture of their customers. The big objective, in many cases, is to create predictive models. You might remember the example of U.S. retailer Target, who is now able to very accurately predict when one of their customers will expect a baby. Using big data, Telecom companies can now better predict customer churn; Wal-Mart can predict what products will sell, and car insurance companies understand how well their customers actually drive. Even government election campaigns can be optimized using big data analytics.

2. Understanding and Optimizing Business Processes

Big data is also increasingly used to optimize business processes. Retailers are able to optimize their stock based on predictions generated from social media data, web search trends and weather forecasts. One particular business process that is seeing a lot of big data analytics is supply chain or delivery route optimization. Here, geographic positioning and radio frequency identification sensors are used to track goods or delivery vehicles and optimize routes by integrating live traffic data, etc. HR business processes are also being improved using big data analytics. This includes the optimization of talent acquisition – Moneyball style, as well as the measurement of company culture and staff engagement using big data tools

3. Personal Quantification and Performance Optimization

Big data is not just for companies and governments but also for all of us individually.

We can now benefit from the data generated from wearable devices such as smart watches or smart bracelets. Take the Up band from Jawbone as an example: the armband collects data on our calorie consumption, activity levels, and our sleep patterns. While it gives individuals rich insights, the real value is in analyzing the collective data.

In Jawbone's case, the company now collects 60 years worth of sleep data every night. Analyzing such volumes of data will bring entirely new insights that it can feed back to individual users. The other area where we benefit from big data analytics is finding love - online this is. Most online dating sites apply big data tools and algorithms to find us the most appropriate matches.

4. Improving Healthcare and Public Health

The computing power of big data analytics enables us to decode entire DNA strings in minutes and will allow us to find new cures and better understand and predict disease patterns. Just think of what happens when all the individual data from smart watches and wearable devices can be used to apply it to millions of people and their various diseases. The clinical trials of the future won't be limited by small sample sizes but could potentially include everyone! Big data techniques are already being used to monitor babies in a specialist premature and sick baby unit. By recording and analyzing every heart beat and breathing pattern of every baby, the unit was able to develop algorithms that can now predict infections 24 hours before any physical symptoms appear. That way, the team can intervene early and save fragile babies in an environment where every hour counts. What's more, big data analytics allow us to monitor and predict the developments of epidemics and disease outbreaks. Integrating data from medical records with social media analytics enables us to monitor flu outbreaks in real-time, simply by listening to what people are saying, i.e. "Feeling bubbish today - in bed with a cold".

5. Improving Sports Performance

Most elite sports have now embraced big data analytics. We have the IBM SlamTracker tool for tennis tournaments; we use video analytics that track the performance of every player in a football or baseball game, and sensor technology in sports equipment such as basket balls or golf clubs allows us to get feedback (via smart phones and cloud servers) on our game and how to improve it. Many elite sports teams also track athletes outside of the sporting environment – using smart technology to track nutrition and sleep, as well as social media conversations to monitor emotional wellbeing.

6. Improving Science and Research

Science and research is currently being transformed by the new possibilities big data brings. Take, for example, CERN, the Swiss nuclear physics lab with its Large Hadron Collider, the world's largest and most powerful particle accelerator. Experiments to unlock the secrets of our universe – how it started and works - generate huge amounts of data. The CERN data center has 65,000 processors to analyze its 30 petabytes of data.

However, it uses the computing powers of thousands of computers distributed across 150 data centers worldwide to analyze the data. Such computing powers can be leveraged to transform so many other areas of science and research.

7. Optimizing Machine and Device Performance Big data analytics help machines and devices become smarter and more autonomous.

For example, big data tools are used to operate Google's self-driving car. The Toyota Prius is fitted with cameras, GPS as well as powerful computers and sensors to safely drive on the road without the intervention of human beings. Big data tools are also used to optimize energy grids using data from smart meters. We can even use big data tools to optimize the performance of computers and data warehouses.

8. Improving Security and Law Enforcement.

Big data is applied heavily in improving security and enabling law enforcement. I am sure you are aware of the revelations that the National Security Agency (NSA) in the U.S. uses big data analytics to foil terrorist plots (and maybe spy on us). Others use big data techniques to detect and prevent cyber attacks. Police forces use big data tools to catch criminals and even predict criminal activity and credit card companies use big data use it to detect fraudulent transactions.

9. Improving and Optimizing Cities and Countries

Big data is used to improve many aspects of our cities and countries. For example, it allows cities to optimize traffic flows based on real time traffic information as well as social media and weather data. A number of cities are currently piloting big data analytics with the aim of turning themselves into Smart Cities, where the transport infrastructure and utility processes are all joined up. Where a bus would wait for a delayed train and where traffic signals predict traffic volumes and operate to minimize jams.

10. Financial Trading

My final category of big data application comes from financial trading. High-Frequency Trading (HFT) is an area where big data finds a lot of use today. Here, big data algorithms are used to make trading decisions. Today, the majority of equity trading now takes place via data algorithms that increasingly take into account signals from social media networks and news websites to make, buy and sell decisions in split seconds.

A Convergence of Key Trends

Steve Lucas, is the Global Executive Vice President and General Manager, SAP Database & Technology at SAP. He's an experienced player in the Big Data analytics space, and we're delighted that he agreed to share some of his insights with us. First of all, according to Lucas, it's important to remember that big companies have been collecting and storing large amounts of data for a long time. From his perspective, the difference between "Old Big Data" and "New Big Data" is accessibility. Here's a brief summary of our interview:

Companies have always kept large amounts of information. But until recently, they stored most of that information on tape. While it's true that the amount of data in the world keeps growing, the real change has been in the ways that we access that data and use it to create value.

Today, you have technologies like Hadoop, for example, that make it functionally practical to access a tremendous amount of data, and then extract value from it. The availability of lower-cost hardware makes it easier and more feasible to retrieve and process information, quickly and at lower costs than ever before.

So it 's the convergence of several trends—more data and less expensive, faster hardware—that 's driving this transformation. Today, we've got raw speed at an affordable price. That cost/benefit has really been a game changer for us.

That's first and foremost—raw horsepower. Next is the ability to do that real-time analysis on very complex sets of data and models, so it 's not just let me look at my financials or let me look at marketing information. And finally, we now have the ability to find solutions for very complex problems in real time.

We asked Steve Lucas to offer some examples of scenarios in which the ability to analyze Big Data in real time is making an impact. Here 's what he told us:

A perfect example would be insurance companies. They need to know the answers to questions like this: As people age, what kinds of different services will they need from us?

In the past, the companies would have been forced to settle for general answers. Today, they can use their data to find answers that are more specific and significantly more useful. Here are some examples that Lucas shared with us from the insurance and retail industries:

You don't have to guess. You can look at actual data, from real customers. You can extract and analyze every policy they've ever held. The answers to your questions are buried in this kind of massive mound of data—potentially petabytes worth of data if you consider all of your insurance customers across the lifespan of their policies.

It 's unbelievable how much information exists. But now you've got to go from the level of petabytes and terabytes down to the level of a byte. That's a very complex process. But today you can do it—you can compare one individual to all the other people in an age bracket and perform an analysis, in real time. That 's pretty powerful stuff. Imagine if a customer service rep had access to that kind of information in real time. Think of all the opportunities and advantages there would be, for the company and for the customer.

Here's another example: You go into a store to buy a pair of pants. You take the pants up to the cash register and the clerk asks you if you would like to save 10 percent off your purchase by signing up for the store 's credit card.

99.9 percent of the time, you 're going to say "no." But now let 's imagine if the store could automatically look at all of my past purchases and see what other items I bought when I came in to buy a pair of pants—and then offer me 50 percent off a similar purchase? Now that would be relevant to me. The store isn 't offering me another lame credit card—it 's offering me something that I probably want, at an attractive price.

The two scenarios described by Lucas aren 't fantasies. Yesterday, the cost of real-time data analysis was prohibitive. Today, real-time analytics have become affordable. As a result, market-leading companies are already using Big Data Analytics to improve sales revenue, increase profits, and do a better job of serving customers.

Before moving on, it 's worth repeating that not all new Big Data technology is open source. For example, SAP successfully entered the Big Data market with SAP HANA, an in-memory database platform for real-time analytics and applications. Products like SAP HANA are reminders that suppliers of proprietary solutions, such as SAP, SAS, Oracle, IBM, and Teradata, are playing—and will obviously continue to play—significant roles in the evolution of Big Data analytics.

Unstructured Data

Unstructured data is information that either does not have a predefined data model and/or does not fit well into a relational database. Unstructured information is typically text heavy, but may contain data such as dates, numbers, and facts as well. The term semi-structured data is used to describe structured data that does not fit into a formal structure of data models. However, semi-structured data does contain tags that separate semantic elements, which includes the capability to enforce hierarchies within the data.

The amount of data (all data, everywhere) is doubling every two years. Most new data is unstructured. Specifically, unstructured data represents almost 80 percent of new data, while structured data represents only 20 percent. Unstructured data tends to grow exponentially, unlike structured data, which tends to grow in a more linear fashion.

Unstructured data is vastly underutilized.

Mining Unstructured Data

Many organizations believe that their unstructured data stores include information that could help them make better business decisions. Unfortunately, it's often very difficult to analyze unstructured data. To help with the problem, organizations have turned to a number of different software solutions designed to search unstructured data and extract important information. The primary benefit of these tools is the ability to glean actionable information that can help a business succeed in a competitive environment.

Because the volume of unstructured data is growing so rapidly, many enterprises also turn to technological solutions to help them better manage and store their unstructured data. These can include hardware or software solutions that enable them to make the most efficient use of their available storage space.

Unstructured Data and Big Data

As mentioned above, unstructured data is the opposite of structured data. Structured data generally resides in a relational database, and as a result, it is sometimes called "relational data." This type of data can be easily mapped into pre-designed fields. For example, a database designer may set up fields for phone numbers, zip codes and credit card numbers that accept a certain number of digits. Structured data has been or can be placed in fields like these. By contrast, unstructured data is not relational and doesn't fit into these sorts of pre-defined data models.

In addition to structured and unstructured data, there's also a third category: semi-structured data. Semi-structured data is information that doesn't reside in a relational database but that does have some organizational properties that make it easier to analyze. Examples of semi-structured data might include XML documents and NoSQL databases.

The term "big data" is closely associated with unstructured data. Big data refers to extremely large datasets that are difficult to analyze with traditional tools. Big data can include both structured and unstructured data, but IDC estimates that 90 percent of big data is unstructured data. Many of the tools designed to analyze big data can handle unstructured data.

Implementing Unstructured Data Management

Organizations use a variety of different software tools to help them organize and manage unstructured data. These can include the following:

Big data tools: Software like Hadoop can process stores of both unstructured and structured data that are extremely large, very complex and changing rapidly.

Business intelligence software: Also known as BI, this is a broad category of analytics, data mining, dashboards and reporting tools that help companies make sense of their structured and unstructured data for the purpose of making better business decisions. □

Data integration tools: These tools combine data from disparate sources so that they can be viewed or analyzed from a single application. They sometimes include the capability to unify structured and unstructured data.

Document management systems: Also called "enterprise content management systems," a DMS can track, store and share unstructured data that is saved in the form of document files.

Information management solutions: This type of software tracks structured and unstructured enterprise data throughout its lifecycle.

Search and indexing tools: These tools retrieve information from unstructured data files such as documents, Web pages and photos.

Industry Examples of Big Data

10 industry verticals that are using big data, industry-specific challenges that these industries face, and how big data solves these challenges.

A study of 16 projects in 10 top investment and retail banks shows that the challenges in this industry include: securities fraud early warning, tick analytics, card fraud detection, archival of audit trails, enterprise credit risk reporting, trade visibility, customer data transformation, social analytics for trading, IT operations analytics, and IT policy compliance analytics, among others.

The Securities Exchange Commission (SEC) is using big data to monitor financial market activity. They are currently using network analytics and natural language processors to catch illegal trading activity in the financial markets.

Retail traders, Big banks, hedge funds and other so-called 'big boys' in the financial markets use big data for trade analytics used in high frequency trading, pre-trade decision-support analytics, sentiment measurement, Predictive Analytics etc.

Since consumers expect rich media on-demand in different formats and in a variety of devices, some big data challenges in the communications, media and entertainment industry include:

- Collecting, analyzing, and utilizing consumer insights
- Leveraging mobile and social media content
- Understanding patterns of real-time, media content usage

Applications of big data in the Communications, media and entertainment industry

Organizations in this industry simultaneously analyze customer data along with behavioral data to create detailed customer profiles that can be used to:

- Create content for different target audiences
- Recommend content on demand

Measure content performance

A case in point is the Wimbledon Championships (YouTube Video) that leverages big data to deliver detailed sentiment analysis on the tennis matches to TV, mobile, and web users in real-time.

Spotify, an on-demand music service, uses Hadoop big data analytics, to collect data from its millions of users worldwide and then uses the analyzed data to give informed music recommendations to individual users.

Amazon Prime, which is driven to provide a great customer experience by offering, video, music and Kindle books in a one-stop shop also heavily utilizes big data.

Big Data Providers in this industry include: Infochimps, Splunk, Pervasive Software, and Visible Measures

Healthcare Providers

The healthcare sector has access to huge amounts of data but has been plagued by failures in utilizing the data to curb the cost of rising healthcare and by inefficient systems that stifle faster and better healthcare benefits across the board.

This is mainly due to the fact that electronic data is unavailable, inadequate, or unusable. Additionally, the healthcare databases that hold health-related information have made it difficult to link data that can show patterns useful in the medical field.

Other challenges related to big data include: the exclusion of patients from the decision making process, and the use of data from different readily available sensors.

Applications of big data in the healthcare sector

Some hospitals, like Beth Israel, are using data collected from a cell phone app, from millions of patients, to allow doctors to use evidence-based medicine as opposed to administering several medical/lab tests to all patients who go to the hospital. A battery of tests can be efficient but they can also be expensive and usually ineffective.

Free public health data and Google Maps have been used by the University of Florida to create visual data that allows for faster identification and efficient analysis of healthcare information, used in tracking the spread of chronic disease.

Education

From a technical point of view, a major challenge in the education industry is to incorporate big data from different sources and vendors and to utilize it on platforms that were not designed for the varying data.

From a practical point of view, staff and institutions have to learn the new data management and analysis tools.

On the technical side, there are challenges to integrate data from different sources, on different platforms and from different vendors that were not designed to work with one another.

Politically, issues of privacy and personal data protection associated with big data used for educational purposes is a challenge.

Manufacturing and Natural Resources

Increasing demand for natural resources including oil, agricultural products, minerals, gas, metals, and so on has led to an increase in the volume, complexity, and velocity of data that is a challenge to handle.

Similarly, large volumes of data from the manufacturing industry are untapped. The underutilization of this information prevents improved quality of products, energy efficiency, reliability, and better profit margins.

Applications of big data in manufacturing and natural resources

In the natural resources industry, big data allows for predictive modeling to support decision making that has been utilized to ingest and integrate large amounts of data from geospatial data, graphical data, text and temporal data. Areas of interest where this has been used include; seismic interpretation and reservoir characterization.

Government

In governments the biggest challenges are the integration and interoperability of big data across different government departments and affiliated organizations.

Applications of big data in Government

In [public services](#), big data has a very wide range of applications including: energy exploration, financial market analysis, fraud detection, health related research and environmental protection.

Some more specific examples are as follows:

Big data is being used in the analysis of large amounts of social disability claims, made to the Social Security Administration (SSA), that arrive in the form of unstructured data. The analytics are used to process medical information rapidly and efficiently for faster decision making and to detect suspicious or fraudulent claims.

The Food and Drug Administration (FDA) is using big data to detect and study patterns of food-related illnesses and diseases. This allows for faster response which has led to faster treatment and less death.

Insurance

Lack of personalized services, lack of personalized pricing and the lack of targeted services to new segments and to specific market segments are some of the main challenges.

In a survey conducted by Marketforce challenges identified by professionals in the insurance industry include underutilization of data gathered by loss adjusters and a hunger for better insight.

Applications of big data in the insurance industry

Big data has been used in the industry to provide customer insights for transparent and simpler products, by analyzing and predicting customer behavior through data derived from social media, GPS-enabled devices and CCTV footage. The big data also allows for better customer retention from insurance companies.

When it comes to claims management, predictive analytics from big data has been used to offer faster service since massive amounts of data can be analyzed especially in the underwriting stage. Fraud detection has also been enhanced.

Retail and Whole sale trade

From traditional brick and mortar retailers and wholesalers to current day e-commerce traders, the industry has gathered a lot of data over time. This data, derived from customer loyalty cards, POS scanners, RFID etc. is not being used enough to improve customer experiences on the whole. Any changes and improvements made have been quite slow.

Applications of big data in the Retail and Wholesale industry

Big data from customer loyalty data, POS, store inventory, local demographics data continues to be gathered by retail and wholesale stores.

In New York's Big Show retail trade conference in 2014, companies like Microsoft, Cisco and [IBM](#) pitched the need for the retail industry to utilize big data for analytics and for other uses including:

- Optimized staffing through data from shopping patterns, local events, and so on
- Reduced fraud
- Timely analysis of inventory

Transportation

In recent times, huge amounts of data from location-based social networks and high speed data from telecoms have affected travel behavior. Regrettably, research to understand travel behavior has not progressed as quickly.

In most places, transport demand models are still based on poorly understood new social media structures.

Applications of big data in the transportation industry

Some applications of big data by governments, private organizations and individuals include:

- Governments use of big data: traffic control, route planning, intelligent transport systems, congestion management (by predicting traffic conditions)
- Private sector use of big data in transport: revenue management, technological enhancements, logistics and for competitive advantage (by consolidating shipments and optimizing freight movement)

Energy and Utilities

The image below shows some of the main challenges in the energy and utilities industry.

Applications of big data in the energy and utilities industry

Smart meter readers allow data to be collected almost every 15 minutes as opposed to once a day with the old meter readers. This granular data is being used to analyze consumption of utilities better which allows for improved customer feedback and better control of utilities use.

In utility companies the use of big data also allows for better asset and workforce management which is useful for recognizing errors and correcting them as soon as possible before complete failure is experienced.

Web Analytics

Web analytics is the measurement, collection, analysis and reporting of web data for purposes of understanding and optimizing web usage. Web analytics is not just a tool for measuring web traffic but can be used as a tool for business and market research, and to assess and improve the effectiveness of a web site. The following are the some of the web analytic metrics: Hit, Page view, Visit / Session, First Visit / First Session, Repeat Visitor, New Visitor, Bounce Rate, Exit Rate, Page Time Viewed / Page Visibility Time / Page View Duration, Session Duration / Visit Duration. Average Page View Duration, and Click path etc.

Most people in the online publishing industry know how complex and onerous it could be to build an infrastructure to access and manage all the Internet data within their own IT department. Back in the day, IT departments would opt for a four-year project and millions of dollars to go that route. However, today this sector has built up an ecosystem of companies that spread the burden and allow others to benefit.

Avinash Kaushik believes there is one interesting paradigm shift that the Web mandates, that corporate information officers (CIOs) are and will continue to lose massive amounts of control over data and create large bureaucratic organizations whose only purpose is to support, collect, create, mash data, and be in the business of data “puking.” He believes such CIOs are “losing control in spades”:

One of the interesting things that I had to grapple with as I embraced the Web and moved to the Web is that the primary way in which data gets collected, processed and stored, and accessed is actually at a third party. I did not have servers any more. I did not actually have implementations. I actually had to massively reduce the size of my implementation and data massaging and data serving and data banking team and rather massively expand the team that analyzes the data. This is the psychologically hard thing for me to do. When I was the BI person that’s basically where most of the money of the company went. A little bit then went on analysts.

Why use big data tools to analyse web analytics data?

Web event data is incredibly valuable

It tells you how your customers actually behave (in lots of detail), and how that varies

Between different customers

For the same customers over time. (Seasonality, progress in customer journey)

How behaviour drives value

It tells you how customers engage with you via your website / webapp

How that varies by different versions of your product
How improvements to your product drive increased customer satisfaction and lifetime value
It tells you how customers and prospective customers engage with your different marketing campaigns and how that drives subsequent behaviour
Deriving value from web analytics data often involves very bespoke analytics
The web is a rich and varied space! E.g.

- Bank
- Newspaper
- Social network
- Analytics application
- Government organisation (e.g. tax office)
- Retailer
- Marketplace

For each type of business you'd expect different :
Types of events, with different types of associated data•
Ecosystem of customers / partners with different types of relationships
Product development cycle (and approach to product development)
Types of business questions / priorities to inform how the data is analysed

Web analytics tools are good at delivering the standard reports that are common across different business types...

- Where does your traffic come from e.g.
 - Sessions by marketing campaign / referrer
 - Sessions by landing page

Understanding events common across business types (page views, transactions, 'goals') e.g.

- Page views per session
- Page views per web page
- Conversion rate by traffic source
- Transaction value by traffic source

Capturing contextual data common people browsing the web

- Timestamps
- Referrer data
- Web page data (e.g. page title, URL)
- Browser data (e.g. type, plugins, language)
- Operating system (e.g. type, timezone)
- Hardware (e.g. mobile / tablet / desktop, screen resolution, colour depth)

Social and Affiliate Marketing

Big Data and Marketing

Dan Springer, CEO of Responsys, defines the new school of marketing: “Today’s consumers have changed. They’ve put down the newspaper, they fast forward through TV commercials, and they junk unsolicited email. Why? They have new options that better fit their digital lifestyle. They can choose which marketing messages they receive, when, where, and from whom. They prefer marketers who talk with them, not at them.

New School marketers deliver what today’s consumers want: relevant interactive communication across the digital power channels: email, mobile, social, display and the web.”

Big Data and the New School of Marketing

Dan Springer, CEO of Responsys, defines the new school of marketing: “Today’s consumers have changed. They’ve put down the newspaper, they fast forward through TV commercials, and they junk unsolicited email. Why? They have new options that better fit their digital lifestyle. They can choose which marketing messages they receive, when, where, and from whom. They prefer marketers who talk with them, not at them. New School marketers deliver what today’s consumers want: relevant interactive communication across the digital power channels: email, mobile, social, display and the web.”

Consumers Have Changed. So Must Marketers.

While using a lifecycle model is still the best way to approach marketing, today’s new cross-channel customer is online, offline, captivated, distracted, satisfied, annoyed, vocal, or quiet at any given moment. The linear concept of a traditional funnel, or even a succession of lifecycle “stages,” is no longer a useful framework for planning marketing campaigns and programs.

Today’s cross-channel consumer is more dynamic, informed, and unpredictable than ever. Marketers must be ready with relevant marketing at a moment’s notice. Marketing to today’s cross-channel consumer demands a more nimble, holistic approach, one in which customer behavior and preference data determine the content and timing—and delivery channel—of marketing messages. Marketing campaigns should be cohesive: content should be versioned and distributable across multiple channels. Marketers should collect holistic data profiles on consumers, including channel response and preference data, social footprint/area of influence, and more. Segmentation strategies should now take into account channel preferences.

Marketers can still drive conversions and revenue, based on their own needs, with targeted campaigns sent manually, but more of their marketing should be driven by and sent via preferred channels in response to—individual customer behaviors and events. How can marketers plan for that? Permission, integration, and automation are the keys, along with a more practical lifecycle model designed to make every acquisition marketing investment result in conversion, after conversion, after conversion.

The Right Approach: Cross-Channel Lifecycle Marketing

Cross-Channel Lifecycle Marketing really starts with the capture of customer permission, contact information, and preferences for multiple channels. It also requires marketers to have the right integrated marketing and customer information systems, so that (1) they can have complete understanding of customers through stated preferences and observed behavior at any given time; and (2) they can automate and optimize their programs and processes throughout the customer lifecycle. Once marketers have that, they need a practical framework for planning marketing activities. Let's take a look at the various loops that guide marketing strategies and tactics in the Cross-Channel

Lifecycle Marketing approach: conversion, repurchase, stickiness, win-back, and re permission (see figure) New School of Marketing

The Avon Lady has been doing it for over a century. Tupperware parties made buying plastics acceptable back in the 1940s. Word-of-mouth marketing has been the most powerful form of marketing since before the Internet was an idea in Tim Berners-Lee's mind and well before Mark Zuckerberg ever entered that now-famous Harvard dorm room. It's really just a VERY big Tupperware party. – Greg Doran, Founder and CEO of TipSpring What Berners-Lee's and Zuckerberg's ground-breaking concepts and inventions do for word-of-mouth marketers is provide a backbone to bring proven marketing concepts outside of the living room to a scale never before seen.

The concept of affiliate marketing, or pay for performance marketing on the Internet is often credited to William J. Tobin, the founder of PC Flowers & Gifts. In the early 1990s Tobin was granted patents around the concept of an online business rewarding another site (an affiliate site) for each referred transaction or purchase. Amazon.com launched its own affiliate program in 1996 and middleman affiliate networks like Linkshare and CommissionJunction emerged preceding the 1990s Internet boom, providing the tools and technology to allow any brand to put affiliate marketing practices to use. Today, one would be hard pressed to find a major brand that does not have a thriving affiliate program. Today, industry analysts estimate affiliate marketing to be a \$3 billion industry. It's an industry that largely goes anonymous. Unlike email and banner advertising, affiliate marketing is a behind the scenes channel most consumers are unaware of. In 2012, the emergence of the social web brings these concepts together. What only professional affiliate marketers could do prior to Facebook, Twitter, and Tumblr, now any consumer with a mouse can do.

Above and beyond the removal of barriers the social web brings to affiliate marketing, it also brings into the mix the same concepts behind the Avon Lady and Tupperware party—product recommendations from a friend network. As many detailed studies have shown, most people trust a recommendation from the people they know. While professional affiliate marketing sites provide the aggregation of many merchant offers on one centralized site, they completely lack the concept of trusted source recommendations. Using the backbone and publication tools created by companies like Facebook and Twitter, brands will soon find that rewarding their own consumers for their advocacy is a required piece of their overall digital marketing mix. What's old is new again. While not every company in the past had the resources or know how to build an army of Avon Ladies, today there is no excuse. The tools are available to them all and

the scale is exponentially larger than ever before. Anyone can recommend a product through the click of a mouse. No more parties needed.

Empowering Marketing with Social Intelligence We also spoke with Niv Singer, Chief Technology Officer at Tracx, a social media intelligence software provider. Niv had quite a bit to say about the big data challenges faced in the social media realm and how it 's impacting the way business is done today—and in the future. As a result of the growing popularity and use of social media around the world and across nearly every demographic, the amount of user-generated content—or “big data”—created is immense, and continues growing exponentially. Millions of status updates, blog posts, photographs, and videos are shared every second.

Successful organizations will not only need to identify the information relevant to their company and products—but also be able to dissect it, make sense of it, and respond to it—in real time and on a continuous basis, drawing business intelligence—or insights—that help predict likely future customer behavior. And if that sounds like a tall and complex order, that 's because it is. Singer explains how this can be a challenging: It can sometimes be a real challenge to unify social profiles for a single user who may be using different names or handles on each of their social networks, so we 've built an algorithm that combs through key factors including content of posts, and location, among others, to provide a very robust identity unification.

This brings us to the topic of influence and the age old debate of “who is an influencer?” To some brands, influence is measured purely by reach and to others, true influence is more of a function of quality and thoughtfulness of posts showing a real understanding of a given topic, and yet others gauge influence via social engagement or conversations. Because influence is so subjective, Singer believes the client should have the flexibility to sort influencers by any of these characteristics:

Very intelligent software is required to parse all that social data to define things like the sentiment of a post. We believe using a system that 's also able to learn over time what that sentiment means to a specific client or brand and then represent that data with increased levels of accuracy provides clients a way to “train” a social platform to measure sentiment more closely to the way they would be doing it manually themselves. We also know it 's important for brands to be able to understand the demographic information of the individual driving social discussions around their brand such as gender, age, and geography so they can better understand their customers and better target campaigns and programs based on that knowledge.

In terms of geography, Singer explained that they are combining social check-in data from Facebook, Foursquare, and similar social sites and applications over maps to show brands at the country, state/region, state, and down to the street level where conversations are happening about their brand, products, or competitors. This capability enables marketers with better service or push coupons in real time, right when someone states a need, offering value, within steps from where they already are, which has immense potential to drive sales and brand loyalty.

These challenges are in the forefront of technology, but also require very creative people and solutions. Every component in the system must be able to be distributed across multiple servers that don't rely on each other. No single point of failure is allowed—the data must therefore be replicated and stored on different machines, but should still be consistent. The data is later accessed in unpredictable ways. Singer likes to use an analogy to a book in a library:

Finding a book by title or ISBN number is easy, even in a very big library. Finding, or counting, all the books written by specific authors is also relatively easy. It gets a little more complicated when we try to locate all the books written in a certain year, since we usually keep the books on shelves and sort them according to the author. If we need to count the number of books that contain the word “data” in their title written every year, it gets even more complicated. . . and when we need to locate all the books that contain the phrase “big data” in them, well, you can imagine.

Fundamentally, Singer doesn't view social data as a silo and, instead, believes that the real power comes in mining social data for business intelligence, not only for marketing, but also for customer support and sales. As a result, they've created a system from the ground up that was architected to be open. It's designed to be a data management system that just happens to be focused on managing unstructured social data, but we can easily integrate with other kinds of data sets. It was built with the expectation that social data would not live in an island, but would be pushed out to other applications to provide added business value and insights and that they would be pulling external data in.

This open approach like Singer is suggesting is extremely important because it enables businesses to take action with the data! Examples include integration with CRM systems like Salesforce.com and Microsoft Dynamics to enable companies to get a more holistic view of what's going with their clients by supplementing existing data sets that can be more static in nature with the social data set, which is more dynamic and real-time. Another example is integration with popular analytics platforms like Google Analytics and Omniture, so marketers can see a direct correlation and payoff of social campaigns through improved social sentiment or an increase in social conversations around their brand or product.

Where does Singer think this is all headed next? To the next big holy grail: an ability to take all this unstructured data and identify a customer's intent to buy. Customer intent is the big data challenge we're focused on solving. By applying intelligent algorithms and complex logic with very deep, real-time text analysis, we're able to group customers in to buckets such as awareness, opinion, consideration, preference and purchase. That ability lets marketers create unique messages and offers for people along each phase of the purchase process and lets sales more quickly identify qualified sales prospects.

One of Tracx customers is Attention, a heavily data-driven social media marketing agency also based in NYC. The Attention team uses the platform as the backbone of their social market research. Attention's CEO and Founder, Curtis Hougland, had this to say about Big Data's impact on marketing: Social media is the world's largest and purest focus group.

Marketers now have the opportunity to mine social conversations for purchase intent and brand lift through Big Data. So, marketers can communicate with consumers when they are emotionally engaged, regardless of the channel. Since this data is captured in real-time, Big Data is coercing marketing organizations into moving more quickly to optimize media mix and message as a result of these insights.

Since this data sheds light on all aspects of consumer behavior, companies are eliminating silos within the organization to align data to insight to prescription across channels, across media, and across the path to purchase. The days of Don Draper are over, replaced by a union of creative and quant. The capability to understand a customer's intent that Hougland and Singer are referring to is not only good for the corporations; it's also a helpful capability for the client too. Think about it, most people communicate socially because they are looking to share, complain, or find something they need. Wouldn't it be great if someone was listening and knew your intent so that they can provide customer assistance or get you what you need for the best price?

Fraud and Big Data

Fraud is intentional deception made for personal gain or to damage another individual. One of the most common forms of fraudulent activity is credit card fraud. The credit card fraud rate in United States and other countries is increasing. As per Javelin's research, "8th Annual Card Issuers' Safety Scorecard: Proliferation of Alerts Lead to Quicker Detection Time and Lower Fraud Costs," credit card fraud incidence increased 87 percent in 2011 culminating in an aggregate fraud loss of \$6 billion. 2

However, despite the significant increase in incidence, total cost of credit card fraud increased only 20 percent. The comparatively small rise in total cost can be attributed to an increasing sophistication of fraud detection mechanisms. According to the Capgemini Financial Services Team: Even though fraud detection is improving, the rate of incidents is rising. This means banks need more proactive approaches to prevent fraud. While issuers' investments in detection and resolution has resulted in an influx of customerfacing tools and falling average detection times among credit card fraud victims, the rising incidence rate indicates that credit card issuers should prioritize preventing fraud.

Social media and mobile phones are forming the new frontiers for fraud. Despite warnings that social networks are a great resource for fraudsters, consumers are still sharing a significant amount of personal information frequently used to authenticate a consumer's identity. Those with public profiles (those visible to everyone) were more likely to expose this personal information. According to Javelin's "2012 Identity Fraud Report: Social Media and Mobile Forming the New Fraud Frontier," 68 percent of

people with public social media profiles shared their birthday information (with 45 percent sharing month, date, and year); 63 percent shared their high school name; 18 percent shared their phone number; and 12 percent shared their pet's name—all are prime examples of personal information a company would use to verify your identity. In order to prevent the fraud, credit card transactions are monitored and checked in near real time. If the checks identify pattern inconsistencies and suspicious activity, the transaction is identified for review and escalation.

The Capgemini Financial Services team believes that due to the nature of data streams and processing required, Big Data technologies provide an optimal technology solution based on the following three Vs:

1. High volume. Years of customer records and transactions (150 billion records per year)
2. High velocity. Dynamic transactions and social media information
3. High variety. Social media plus other unstructured data such as customer emails, call center conversations, as well as transactional structured data Capgemini's new fraud Big Data initiative focuses on flagging the suspicious credit card transactions to prevent fraud in near real-time via multi-attribute monitoring. Real-time inputs involving transaction data and customers records are monitored via validity checks and detection rules. Pattern recognition is performed against the data to score and weight individual transactions across each of the rules and scoring dimensions.

A cumulative score is then calculated for each transaction record and compared against thresholds to decide if the transaction is potentially suspicious or not. The Capgemini team pointed out that they use an open-source weapon named Elastic Search, which is a distributed, free/open-source search server based on Apache Lucene (see Figure 2.2). It can be used to search all kind of documents at near real-time. They use the tool to index new transactions, which are sourced in real-time, which allows analytics to run in a distributed fashion utilizing the data specific to the index. Using this tool, large historical data sets can be used in conjunction with real-time data to identify deviations from typical payment patterns. This Big Data component allows overall historical patterns to be compared and contrasted, and allows the number of attributes and characteristics about consumer behavior to be very wide, with little impact on overall performance.

Once the transaction data has been processed, the percolator query then performs the functioning of identifying new transactions that have raised profiles. Percolator is a system for incrementally processing updates to large data sets. Percolator is the technology that Google used in building the index—that links keywords and URLs—used to answer searches on the Google page.

Percolator query can handle both structured and unstructured data. This provides scalability to the event processing framework, and allows specific suspicious transactions to be enriched with additional unstructured information—phone location/geospatial records, customer travel schedules, and so on.

This ability to enrich the transaction further can reduce false positives and increase the experience of the customer while redirecting fraud efforts to actual instances of suspicious activity.

Another approach to solving fraud with Big Data is social network analysis (SNA). SNA is the precise analysis of social networks. Social network analysis views social relationships and makes assumptions. SNA could reveal all individuals involved in fraudulent activity, from perpetrators to their associates, and understand their relationships and behaviors to identify a bust out fraud case.

According to a recent article in bankersonline.com posted by Experian, “bust out” is a hybrid credit and fraud problem and the scheme is typically defined by the following behavior:

- The account in question is delinquent or charged-off.
- The balance is close to or over the limit.
- One or more payments have been returned.
- The customer cannot be located.
- The above conditions exist with more than one account and/or financial institution.

There are some Big Data solutions in the market like SAS 's SNA solution, which helps institutions and goes beyond individual and account views to analyze all related activities and relationships at a network dimension. The network dimension allows you to visualize social networks and see previously hidden connections and relationships, which potentially could be a group of fraudsters. Obviously there are huge reams of data involved behind the scene, but the key to SNA solutions like SAS 's is the visualization techniques for users to easily engage and take action.

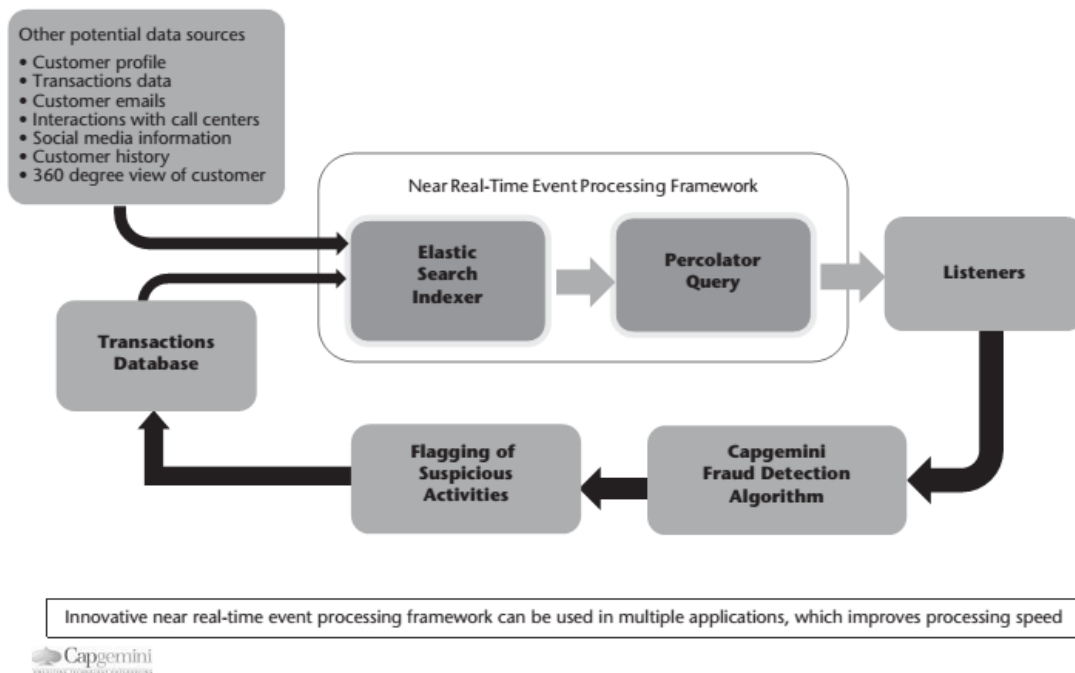


Figure 2.2 Fraud Detection Powered by Near Real-Time Event Processing Framework
Source: © 2012 Capgemini.

Risk and Big Data

Many of the world's top analytics professionals work in risk management. It would be an understatement to say that risk management is data-driven—without advanced data analytics, modern risk management would simply not exist. The two most common types of risk management are credit risk management and market risk management. A third type of risk, operational risk management, isn't as common as credit and market risk.

The tactics for risk professionals typically include avoiding risk, reducing the negative effect or probability of risk, or accepting some or all of the potential consequences in exchange for a potential upside gain. Credit risk analytics focus on past credit behaviors to predict the likelihood that a borrower will default on any type of debt by failing to make payments which they obligated to do. For example, "Is this person likely to default on their \$300,000 mortgage?"

Market risk analytics focus on understanding the likelihood that the value of a portfolio will decrease due to the change in stock prices, interest rates, foreign exchange rates, and commodity prices. For example, “Should we sell this holding if the price drops another 10 percent?”

Credit Risk Management

Credit risk management is a critical function that spans a diversity of businesses across a wide range of industries. Ori Peled is the American Product Leader for MasterCard Advisors Risk & Marketing Solutions. He brings several years of information services experience in his current role with MasterCard and having served in various product development capacities at Dun & Bradstreet. Peled shares his insight with us on credit risk:

Whether you're a small B2B regional plastics manufacturer or a large global consumer financial institution, the underlying credit risk principles are essentially the same: driving the business using the optimal balance of risk and reward. Traditionally, credit risk management was rooted in the philosophy of minimizing losses. However, over time, credit risk professionals and business leaders came to understand that there are acceptable levels of risk that can boost profitability beyond what would normally have been achieved by simply focusing on avoiding write-offs. The shift to the more profitable credit risk management approach has been aided in large part to an ever-expanding availability of data, tools, and advanced analytics.

Credit risk professionals are stakeholders in key decisions that address all aspects of a business, from finding new and profitable customers to maintaining and growing relationships with existing customers. Maximizing the risk and reward opportunities requires that risk managers understand their customer portfolio, allowing them to leverage a consistent credit approach while acknowledging that you can't treat every customer the same. As businesses grow, what starts out as a manual and judgmental process of making credit decisions gives way to a more structured and increasingly automated process in which data-driven decisions becomes the core. Decisions that impact not only revenue but also operational costs like staffing levels of customer support representatives or collections agents.

The vast amount of both qualitative and quantitative information available to credit risk professionals can be overwhelming to digest and can slow down a process with potential sales at risk. With advanced analytical tools, these abundant and complex data sources can be distilled into simple solutions that provide actionable insights and are relatively easy to implement.

As an example, credit scoring solutions allow risk managers to apply a credit policy more efficiently and consistently across the business. Scoring solutions can take various data sources and produce a score that computes the odds of a customer behaving in a specific way.

Traditional scoring methods focus on predicting the likelihood of delinquency or bankruptcy but additional scoring solutions can also help companies identify the profitability potential of customers or from a marketing perspective, the propensity to spend.

Additionally, companies are leveraging and combining multiple analytical solutions at the same time—this could be a combination of proprietary scoring solutions and third party scoring like those provided by specialized analytics providers and the consumer and commercial bureaus (e.g., Experian, Equifax, D&B, etc.). As you look across the credit risk management lifecycle, rich data sources and advanced analytics are instrumental throughout. From a customer acquisition perspective, credit risk managers decide whether to extend credit and how much. Lacking any previous experience with the prospect, they depend heavily on third-party credit reports and scores and may assist marketing organizations in employing customized look-alike models to help identify prospective best customers.

From an existing customer standpoint, the focus shifts to ongoing account management and retaining profitable accounts. This requires periodic customer risk assessments that influence key decisions on credit line increases and decreases. Again, advanced analytical solutions come into play, especially in larger organizations where the volume of accounts dictate a need for automated decisioning solutions that leverage behavior scores and other data sources. Continuous monitoring of an existing portfolio can also help credit risk managers forecast expected losses and better manage their collections efforts. Advanced analytics in the collections phase can help identify customers most likely to pay or even respond to different collection strategies and approaches.

The future of credit risk management will continue to change as we leverage new data sources emanating from a highly digital and mobile world. As an example, social media and cell phone usage data are opening up new opportunities to uncover customer behavior insights that can be used for credit decisioning. This is especially relevant in the parts of the world where a majority of the population is unbanked and traditional bureau data is unavailable.

As Figure 2.3 illustrates, there are four critical parts of the typical credit risk framework: planning, customer acquisition, account management, and collections. All four parts are handled in unique ways through the use of Big Data.

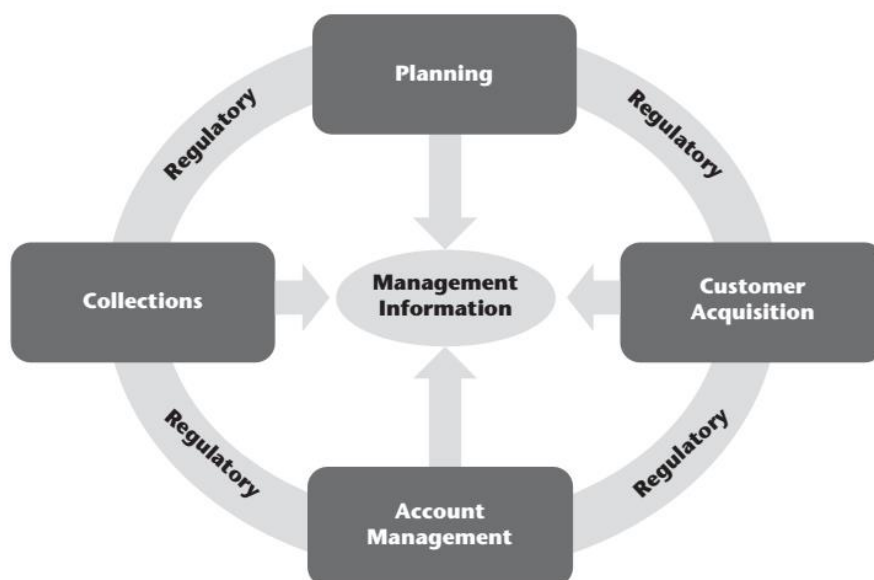


Figure 2.3 Credit Risk Framework
 Source: Ori Peled.

Big Data and Algorithmic Trading

Partha Sen is the CEO of Fuzzy Logix, a company that specializes in highperformance, cross platform in database, and GPU (graphics processing unit) analytics. Sen spent over 15 years as a quantitative analyst in the financial services industry. Over the course of his career, he developed over 700 highly parallelized algorithms in C/C++.

He, along with a team of very talented quantitative professionals, now leverages his formidable expertise to help customers across a number of industries. Sen has seen a significant shift in the use of data in the financial services industry over the past decade. “Financial institutions,” he says, “particularly investment banks, have been at the forefront of applying analytics for risk management, proprietary trading, and portfolio management.”

As most of you know, many investment banks use algorithmic trading, a highly sophisticated set of processes in which “insights” are made “actionable” via automated “decisions.” Algorithmic trading relies on sophisticated mathematics to determine buy and sell orders for equities, commodities, interest rate and foreign exchange rates, derivatives, and fixed income instruments at blinding speed.

A key component of algorithmic trading is determining return and the risk of each potential trade, and then making a decision to buy or sell. Quantitative risk analysts help banks develop trading rules and implement these rules using modern technology. Algorithmic trading involves a huge number of transactions with complex interdependent data, and every millisecond matters.

It's fair to say that these days banks focus more closely on market risk today than ever before. Market risk is basically the risk due to a fluctuation in the value of assets in the marketplace. For a given portfolio, you are trying to determine the probability that the value of the portfolio will fall within a certain threshold within five days, within seven days, within one month.

With asset volatilities as high as they have been observed in the last few years, a lot of stress is being put on market risk. Sophisticated methods for managing market risk depend very heavily of modern technology. Apart from investment banks, corporate and retail banks also rely very heavily on quantitative techniques. Two areas that readily come to mind are marketing, where they solicit households for financial products like credit cards, and credit risk management, where banks try to understand the probability that borrowers will default on loan. The models used in these areas for future outcomes are created with huge number of variables.

For example, a model of the default risk for credit cards could be influenced by demographic factors, whether people have a job or not, what is the growth in the economy, and interest rates. There can be hundreds of factors or variables for each credit card. A typical retail bank will evaluate somewhere north of 5,000 factors for one given model to establish or calculate the probability of each of the borrowers defaulting. The number of calculations just for the risk factor can easily climb into billions of calculations being performed to calculate risk for a portfolio. Crunching Through Complex Interrelated Data

In a frictionless economy, time is the critical driver to gain and sustain a competitive advantage. Every second, or more correctly, every millisecond counts today. Banks have graduated from daily evaluation of risk to intra-day risk evaluations. "Risk management on a daily basis is a thing of the past because there are higher volatilities," says Sen of the marketplace today.

"You can see the volatilities unfold in the marketplace when there is an event or an announcement about macro events—unemployment rate or interest rate going up or down or important geo-political events.

News often causes uncertainty in the minds of investors and thus volatilities in financial markets increase. When volatility goes up during the course of a day or trading session, it has an instantaneous effect on the value of financial instruments." For market risk, the data explodes very quickly. Today, the portfolios being evaluated are quite large and include multiple financial instruments. For example, an investment bank will have a portfolio of equities, along with a portfolio of options—both calls and puts on equities.

In addition, there will be foreign exchange trades, a portfolio of interest rate instruments, and interest rate derivatives. Some banks may have more complex products in their portfolios like exotic options—Bermudans, Asian options, digital options, and such.

An often used convention, according to Sen, is to calculate the mark-to market value of the underlying financial instruments and thus calculate the risks. To show how this works, he gave us this example:

Let 's say that you have an investment bank that has an equity derivative in its portfolio and the value of this derivative will change in the future. That change is going to be influenced by the spot price of the underlying stock, the volatility of that stock, interest rate, and time to maturity.

The convention is that every day you take the value of that derivative and you perform scenario analysis over a time horizon—the next 30 days—to determine what will be the value. Will it be \$3.00 instead of \$10.00 that the bank has on the books? Or will it be \$13.00? In this type of scenario analysis, you create multiple scenarios and price the derivative against all the scenarios. Even for a single instrument it is possible to have hundreds of thousands of scenarios. Naturally, when you have hundreds of thousands of equity derivatives in your portfolio different equities, different maturities, and different strike prices, the problem of scenario analysis becomes very complex.

Intraday Risk Analytics, a Constant Flow of Big Data To maintain competitive advantage, banks need to continuously evaluate their models, including the performance of the production models, and also continuously try to build new models to incorporate new variables with new and evolving macroeconomic conditions in a faster way. Banks have also moved from daily risk management to intraday risk management. Intraday risk management involves pricing the entire portfolio and calculating the risk limits of each of the counter-parties within the bank 's portfolio. The problem gets very complex and computationally intensive.

Let 's take an example of intraday risk evaluation of equities. The potential changes within a day include the spot price, the volatility of the underlying equity, and the risk free rate. If we do some basic scenario analysis—say 100 risk-free rate scenarios that could manifest themselves during the course of the day—that means calculating 100 scenarios for the spot price of the equity during the course of the day, 100 scenarios for volatility during the course of the day, and 100 scenarios for risk-free rate during the course of the day.

For the bank to do their basic scenario analysis, it takes a million calculations for determining the value at risk for just that one instrument. And it must happen fast enough so that risk limits on the entire portfolio can be evaluated several times during the course of the day “The only option that I can currently see,” says Sen, “is to be solving these problems using a very large amount of parallelized computations and that is only possibly doable with GPUs.

Using this type of high performance compute technology, we can determine value at risk for 100 million scenarios in less than ten milliseconds using just one of these GPU cards. The real power comes into play when you use multiple cards and parallelize the entire workload. That 's when you can do scenario analysis across your entire portfolio in about 15 minutes.”

Calculating Risk in Marketing

While risk analytics is used for risk management, banks are using risk predictive analytics for marketing as well. For example, when a bank scores its customers and prospects for credit card solicitations, it will use some risk management tools as well. In addition to determining who has a high likelihood of responding to promotional offers, the bank will want to consider the underlying risk for each of the prospects to whom the solicitations are being sent.

Without taking into account risk profiles of individuals, bank promotion responses can result in customers with a higher risk profile. “One of the challenges for retail banks,” according to Sen, “is to score such large numbers of people for its marketing initiatives” Given people 's exact situation, you have to determine what are the right products to promote. Maybe somebody has a home loan but doesn't have a credit card or debit card. In addition, you also have to score your existing customers to determine the borrowers whose probabilities of not paying on their credit card, or on the mortgage, is rising. Once you know who these potential defaulters could be, you can see what you can do to mitigate risk of default. The sheer volume of the population that you have to score compounds the problem. You have to score it quickly because you have to take action promptly be it promotion or risk mitigation.

Other Industries Benefit from Financial Services' Risk Experience Outside of financial services there are other industries that can benefit from this work, such as retail, media, and telecommunications. They are following suit to include evaluation of adverse select in their promotional offers.

While the adoption of analytics has been slower in other industries, momentum is starting to build around Big Data analytics. Marketing is an area that is clearly more mature in terms of adopting analytics in the areas of for marketing campaign management, targeted micromarketing (sending of different offers to different types of people depending on their likelihood to buy), and market basket analysis, which indicates what people buy together and more.

For example, in retail, forecasting is a key area where analytics is being applied. Customer churn analysis has been used by banks to determine who is likely to cancel their credit card or account. This is the same technique that is being used by telecommunication companies and retailers today to determine customer defection. Churn is also a factor used in determining customer lifetime value. Customer lifetime value indicates how much money a firm can make over the customer 's lifetime, that is the period of association of the customer with the firm.

Companies typically use the customer lifetime value to segment their customers and determine which are the customers to focus on. The insurance industry today uses actuarial models for estimating losses.

However, the emerging trend is to use Monte-Carlo simulations for estimating potential losses in insurance portfolios. These computationally complex models require a large footprint of hardware in order to handle the massive calculations. The cost of acquiring and maintaining such hardware sometimes becomes the impediment to adoption of analytics in enterprises. “With the advent of GPU technology, however, that will change,” says Sen. Another use for Big Data analytics in banks is identifying manipulative behavior or fraudulent activities in real-time so that you can mitigate or penalize the behavior immediately. For this, you have to dig through the voluminous transactions and find the patterns quickly. “It ’s always good to catch the culprit but by that time—five years or five days later—a lot of honest players have been disadvantaged.” And what can you do? “Well, not much. says Sen. However, Sen observes, “if you can catch it [the fraud] while it ’s happening, then you can focus on immediate enforcement so the manipulators can ’t twist the prices in the market to negatively impact the retail investor or even the institutional investor, who is a fair player. By quickly catching and correcting this market manipulative behavior you ’re creating a fair trading platform.”

Big Data In Medicine

The new era of data-driven health care is already producing new kinds of heroes. For example, Murali Ramanathan is co-director, Data Intensive Discovery Institute, State University of New York at Buffalo. Ramanathan uses Big Data analytics to identify the genetic variations that predispose individuals to multiple sclerosis (MS). Here ’s a brief description of what he does, in his own words:

I am not a computer scientist, I ’m a pharmaceutical scientist and work on the role of environmental factors, interactions between environmental factors in multiple sclerosis, which is a neurological disease. What we ’ve got are data sets that contain thousands of genes. Typically, our data sets contain somewhere between 100,000 to 500,000 genetic variations.

Our algorithms identify the interactions (between environmental factors and diseases) and they also have rapid search techniques built into them. We also want to be able to do statistical analysis. In our case, we are doing permutation analysis, which can be very, very time consuming if not properly done. With today ’s technology, we can get about 500,000 genetic variations in a single patient sample.

Nate Burns, a colleague of Ramanathan ’s at SUNY Buffalo, paints a vivid description of the challenge facing pioneers of data-intensive quantitative pharmacology:

The data set is very large—a 1,000 by 2,000 matrix. What makes it interesting is when you try to do an interaction analysis for first and second order interactions, basically each of those 500,000 genetic locations is compared to each of all the rest of the 500,000 genetic locations for the first order; then you have to do that twice, so you've cut 500,000 to a third, for second order interactions and so on.

It becomes exceedingly challenging as you move into more interactions. Basically, a second order interaction would be 500,000 squared, a third order would be 500,000 cubed, and so on.

The good news is that results from the MS study can potentially help researchers understand other autoimmune diseases (such as rheumatoid arthritis, diabetes, and lupus) and neurodegenerative diseases such as Parkinson's and Alzheimer's. "MS actually occupies a space between these two categories of diseases," says Ramanathan. "Our goal is finding the similarities and differences by looking at the data sets."

Advertising And Big Data

Let's take one of the oldest business practices, advertising, which dates back to the days when ancient Egyptians used Papyrus to make sales banners to promote their businesses. Back then it was a simple matter of promoting your business through the use of visual promotional advertising. Now let's fast forward to an incredible scene on the streets of Madison Avenue, New York City, during the 1960s.

Every present-day businessperson smirks in either jealousy or disbelief when they see the work-life of the advertising executive character from AMC's drama *Mad Men*, Donald Draper. Draper's character is partially based on Draper Daniels, the creative head of the Leo Burnett advertising agency in 1950s who created the Marlboro Man campaign. As Don said, "Just think about it deeply, then forget it . . . then an idea will jump up in your face." A bunch of executives ideating about the next big idea over Manhattan cocktails and Lucky Strike cigarettes wasn't that far from reality in those days. With a history dating back to 1873, Foote, Cone & Belding Worldwide is one of the oldest providers of advertising and marketing services. Fairfax Cone inherited the agency from Albert Lasker, who can justifiably be called the founder of the modern advertising industry. Together with his colleagues, Emerson Foote, and Don Belding, Cone led the agency for over 30 years. Cone's quote was stated around the same time when companies were using rather primitive sales and marketing tactics to "go see someone." Salesmen were lugging Electrolux Vacuum cleaners from house to house pitching their high-end equipment and competing against manufacturers like Hoover and Oreck. The salesmen had a tough job because the only customer information they had was picking a neighborhood where they felt people could afford their product. The truth is they were not welcome at any home and were greeted with a scowl or totally ignored by people pretending to not be home. There was one key question each salesman would ask that increased their chance of being invited in, "Do you own an Electrolux?" If the answer was yes, the salesman would offer to service their equipment with a tune-up.

This would result in an infrequent upsell to a new vacuum, but most of the time they were lucky if they sold a pack of new bags! While in Great Britain, the firm launched an advertising campaign with the slogan “Nothing sucks like an Electrolux.” This clever slogan was accompanied by a visual of the Leaning Tower of Pisa next to the latest series of the Electrolux vacuum. Since Electrolux is a Scandinavian-based company, most people thought this double entendre was a blunder. However, the slogan was deliberate and designed to have “stopping power,” and it certainly succeeded in grabbing the audience’s attention. The Electrolux vacuum brand sold very well in Great Britain for some time and people still remember the slogan. Although the campaign was rejected in the United States in the late 1960s, some think that this campaign humor would work in America today, especially during the Super Bowl. The early advertising executives had a powerful means to reach their audiences, which was billboard, newspaper, radio, and eventually television. However, their clients were focused on the big idea because they were desperate to get their messages through these channels. As the industry matured, they demanded to learn more about their audiences, which created demand for firms such as Nielsen Media Research, which would statistically measure which television programs are watched by different segments of the population. This would help the advertisers pick the best place to place their ads (media spend). After years of refinement, clever media planning, and the inclusion of more and more data, marketers got pretty savvy at targeting their ads.

Big Data Feeds the Modern-Day Donald Draper

To get a feel for how Big Data is impacting the advertising market, we sat down with Randall Beard, who is currently the global head of Nielsen’s Global Head of Advertiser Solutions. Essentially what Beard’s team does is connect what people watch and what people buy to help their clients optimize advertising and media return on investment. The Nielsen experience is great, but the best part of interviewing Beard is that before Nielsen he actually worked on the client side for 25 years at companies such as the big advertising spender P&G. Needless to say, he knows his stuff.

What’s driving all of this is not just that they’re spending a lot of money but CEOs/CFOs are looking at the chief marketing officers and saying look, if we’re going to spend \$100 million, \$500 million, a billion in advertising and media, show me the money. Show me this stuff works or we’re not going to spend the money on it. There was this huge demand for accountability that’s driving the need for the marketing heads to answer these questions. —Randall Beard

Reach, Resonance, and Reaction

Beard explained that big data is now changing the way advertisers address three related needs:

1. How much do I need to spend? “It’s basic and fundamental but it’s amazing to me that we’re sitting here in 2012 and advertisers still have a really hard time answering the question, How much do I need to spend next year? I know what my revenue goal is next year, and I know how much profit I need to deliver. What do I need to spend to deliver that?”

2. How do I allocate that spend across all the marketing communication touch points? “Answering the question, “how do I allocate my marketing communications spending across paid, owned and earned media is increasingly difficult. If you think about that question, it’s getting harder and harder to answer because of the fragmentation of media, the rise of digital, and the increasing importance of social media. If I’m going to spend money in digital, how much do I allocate to banner versus rich media versus online video versus search? How about mobile, how about apps? It’s further complicated by the fact that all these things work together in a complementary manner. You can’t even think about them as independent things.”

3. How do I optimize my advertising effectiveness against my brand equity and ROI in real-time. “The old paradigm was I go out and run a campaign. Maybe after the fact, I measure it . . . maybe . . . try to determine some ROI then plan for the next cycle of advertising. Basically, advertisers are saying that’s not good enough. I want to know within days, or at least weeks, how my advertising is performing in the market and what levers to pull, what knobs to turn, so that I get a higher ROI.”

Given these needs, advertisers need to be able to measure their advertising end to end. What does this mean?

To start with, they need to identify the people who are most volumetrically responsive to their advertising. And then answer questions such as: What do those people watch? How do I reach them? “With more and more data, and the ability to measure what people watch and buy at the household level, there is the capability to identify those people who were most volumetrically responsive to your advertising.

Then you can figure out: What TV programs do those people watch? What do they do online? How do I develop my media plan against that intended audience? That’s the first part of reach,” explained Beard. Now the second part of the “reach” equation is to understand if you are actually reaching your desired audience. If you think about the online world, it’s a world where you can deliver 100 million impressions but you never really know for sure who your campaign was actually delivered to.

If your intended audience is women aged 18 to 35, of your 100 million impressions, what percentage of impressions were actually delivered to the intended audience? What was the reach, what was the frequency, what was the delivery against the intended audience? For all the great measurement that people can do online, that hasn’t been well measured historically. This is the other part of reach—delivering your ads to the right audience.

Let’s now talk about resonance. If you know whom you want to reach and you’re reaching them efficiently with your media spend, the next question is, are your ads breaking through? Do people know they’re from your brand? Are they changing attitudes? Are they making consumers more likely to want to buy your brand? This is what I call “resonance.”

Lastly, you want to measure the actual behavioral impact. If you've identified the highest potential audience, reached them efficiently with your media plan, delivered ads that broke through the clutter and increased their interest in buying your brand—did it actually result in a purchase? Did people actually buy your product or service based on exposure to your advertising? At the end of the day, advertising must drive a behavioral “reaction” or it isn't really working.

Beard explained the three guiding principles to measurement:

1. End to end measurement—reach, resonance and reaction
2. Across platforms (TV, digital, print, mobile, etc.)
3. Measured in real-time (when possible) The Need to Act Quickly (Real-Time When Possible) When you start executing a campaign, how do you know on a daily basis whether your advertising campaign is actually being delivered to your intended audience the way it's supposed to?

For example, in digital, ad performance will differ across websites. Certain websites are really good; certain websites are really bad. How do you optimize across sites “on the fly?” By moving money out of weak performing sites and into better performing sites. Beard describes how real time optimization works: I'm one week into my new ad campaign. There's good news and bad news. The good news is that my ad is breaking thru and is highly memorable.

The bad news is that consumers think my ad is for my key competitor. I work with my agency over the weekend to edit the spot, and it goes back on air. Presto! Branding scores increase. A week later, I see that of my three ads on air, two have high breakthrough but one is weak. I quickly take the weak performing ad off air and rotate the media spend to the higher performing ads.

Breakthru scores go up! My campaign soon moves from running only: 30's to a mix of: 15's and: 30s, a fairly typical plan. Real time data shows me that my 15s work as well as my 30s. Why spend money on 30s? I move all the weight to 15-second ads—and see scores continue to grow.

In digital, I see that brand recall increases with exposure frequency up to two exposures, and then levels off. My agency caps exposure frequency at two. I use the savings from reduced frequency to buy more sites and extend reach. You have real-time optimization that's going on, which is data driven instead of just gut driven! The measurement tools and capabilities are enabling this and so there's a catch-up happening both in terms of advertising systems and processes, but also just the industry infrastructure to be able to actually enable all of this real-time optimization.”

Measurement Can Be Tricky Beard gave an example of the complexity of measurement. There are tools that allow you to tag digital advertising and, typically, through a panel of some kind, you can read those people who were exposed to the advertising and those who were not and measure their actual offline purchase behavior. In doing this for a large beer client, we could see that this campaign generated (after the fact) a 20 percent sales increase among consumers exposed versus not exposed to the advertising.

You (the average person) would look at that and say, wow, looks pretty good—my advertising is working. But the sales results aren't everything. Beard elaborates on the first part of the end-to-end measurement, involving the reach: When we looked at reach for this particular client, their intended audience was males, aged 21–29. Of their 100 million delivered impressions, only about 40 million were actually delivered to males aged 21–29. Sixty million went to someone other than their intended audience; some went to kids (not good for a beer brand); some went to people 65+. You start out by saying wow, how much better could I have done, if instead of 40% of my impressions hitting my intended audience, I had 70 or 80% of the impressions hitting them.

When you look at the 40 percent of impressions that hit the intended audience, the reach and frequency of those was something like a 10 percent reach and a 65 frequency. In other words, they only hit about 10 percent of their intended audience, but each of these people was bombarded with, on average, 65 ads! That's not quite the optimization one would hope for.

There's a lot of science in advertising that shows that by maximizing reach and minimizing frequency, you get your best response. If they had been measuring all of this in real time, they could have quickly adjusted the plan to increase delivery to the intended audience, increase reach, and reduce frequency. Content Delivery Matters Too Let's now look at ad performance by website.

The ads were on twelve websites: four were terrible; the breakthrough was terrible, branding was terrible—the ads didn't perform well in those sites. The other ones were really good. If they had measured that in flight, they could have moved spending out of the bad performing sites, into good performing sites, and further improved results. Beard explains the importance of end-to-end measurement: When I think about it, it's almost like the reach times resonance equals reaction.

Of course, this isn't arithmetically true, but it illustrates that while measuring the sales impact alone is great, it's not enough. You could have great sales impact and still be completely non-optimized on the reach and resonance factors that caused the reaction." Optimization and Marketing Mixed Modeling Marketing mixed modeling(MMM) is a tool that helps a Marketing mixed modeling(MMM) is a tool that helps advertisers understand the impact of their advertising and other marketing activities on sales results.

MMM can generally provide a solid understanding of the relative performance of advertising by medium (e.g., TV, digital, print, etc.), and in some cases can even measure sales performance by creative unit, program genre, website, and so on. Now, we can also measure the impact on sales in social media and we do that through market mixed modeling. Market mixed modeling is a way that we can take all the different variables in the marketing mix—including paid, owned, and earned media—and use them as independent variables that we regress against sales data and trying to understand the single variable impact of all these different things.

Since these methods are quite advanced, organizations use high-end internal analytic talent and advanced analytics platforms such as SAS or point solutions such as Unica and Omniture.

Alternatively, there are several boutique and large analytics providers like Mu Sigma that supply it as a software-as-a-service (SaaS). MMM is only as good as the marketing data that is used as inputs. As the world becomes more digital, the quantity and quality of marketing data is improving, which is leading to more granular and insightful MMM analyses.

Beard's Take on the Three Big Data Vs in Advertising Beard shared his perspective on how the three Vs (volume, velocity, and variety) have impacted advertising:

Volume In the old days, this is not that old, not even Mad Mends, maybe 20 to 25 years ago, you would copy test your advertising. The agency would build a media plan demographically targeted and you'd go execute it. That was pretty much it. Maybe 6 to 12 months down the road, you'd try to use scanner data or whatever sales data you had to try to understand if there was any impact. In today's world, there is hugely more advertising effectiveness data.

On TV advertising, we can measure every ad in every TV show every day, across about 70 percent of the viewing audience in the U.S. We measure clients digital ad performance hourly—by ad, by site, by exposure, and by audience. On a daily or weekly basis, an advertiser can look at their advertising performance. The volume of information and data that is available to the advertiser has gone up exponentially versus what it was 20 years ago.

Velocity There are already companies that will automate and optimize your advertising on the web without any human intervention at all based on click-thru. It's now beginning to happen on metrics like breakthrough, branding, purchase intent, and things like that. This is sometimes called programmatic buying. Literally, you'll have systems in place that will be measuring the impact of the advertising across websites or different placements within websites, figuring out where the advertising is performing best. It will be automated optimization and reallocation happening in real-time. The volume and the velocity of data, the pace at which you can get the data, make decisions and do things about it is dramatically increased.

Variety Before, you really didn't have a lot of data about how your advertising was performing in market. You have a lot more data and it's a lot more granular. You can look at your brand's overall advertising performance in the market. But you can also decompose it to how much of a performance is due to the creative quality, due to the media weight, how much is due to the program that the ads sit in. How much is due to placement: time of day, time of year, pod position, how much is due to cross-platform exposure, how much is due to competitive activity. Then you have the ability to optimize on most of those things—in real time. And now you can also measure earned (social) and owned media. Those are all things that weren't even being measured before."

BIG DATA TECHNOLOGIES

Technology is radically changing the way data is produced, processed, analyzed, and consumed. On one hand, technology helps evolve new and more effective data sources. On the other, as more and more data gets captured, technology steps in to help process this data quickly, efficiently, and visualize it to drive informed decisions. Now, more than any other time in the short history of analytics, technology plays an increasingly pivotal role in the entire process of how we gather and use data.

Open-source software is computer software that is available in source code form under an open-source license that permits users to study, change, and improve and at times also to distribute the software. The open-source name came out of a 1998 meeting in Palo Alto in reaction to Netscape's announcement of a source code release for Navigator (as Mozilla).

Although the source code is released, there are still governing bodies and agreements in place. The most prominent and popular example is the GNU General Public License (GPL), which "allows free distribution under the condition that further developments and applications are put under the same license." This ensures that the products keep improving over time for the greater population of users. Some other open-source projects are managed and supported by commercial companies, such as Cloudera, that provide extra capabilities, training, and professional services that support open-source projects such as Hadoop. This is similar to what Red Hat has done for the open-source project Linux.

"One of the key attributes of the open-source analytics stack is that it's not constrained by someone else's predetermined ideas or vision," says David Champagne, chief technology officer at Revolution Analytics, a provider of advanced analytics. "The open-source stack doesn't put you into a straitjacket. You can make it into what you want and what you need. If you come up with an idea, you can put it to work immediately.

That's the advantage of the opensource stack—flexibility, extensibility, and lower cost." "One of the great benefits of open source lies in the flexibility of the adoption model: you download and deploy it when you need it," said Yves de Montcheuil, vice president of marketing at Talend, a provider of open-source data integration solutions. "You don't need to prove to a vendor that you have a million dollars in your budget. With open source, you can try it and adopt it at your own pace."

David Smith of Revolution Analytics has written many blogs and papers about the new open-source analytics stack. Smith is vice president of marketing at Revolution Analytics in Palo Alto. He observes that the pace of software development has accelerated dramatically because of open-source software. He follows this observation by describing how this phenomenon is setting the stage for a new "golden age" of software development: In the past, the pace of software development was moderated by a

relatively small set of proprietary software vendors. But there are clear signs that the old software development model is crumbling, and that a new model is replacing it.

The old model's end state was a monolithic stack of proprietary tools and systems that could not be swapped out, modified, or upgraded without the original vendor's support. This model was largely unchallenged for decades. The status quo rested on several assumptions, including:

1. The amounts of data generated would be manageable
 2. Programming resources would remain scarce
 3. Faster data processing would require bigger, more expensive hardware
- Many of those underlying assumptions have now disappeared, David writes: The sudden increase in demand for software capable of handling significantly larger data sets, coupled with the existence of a worldwide community of open-source programmers, has upended the status quo. The traditional analytics stack is among the first "victims" of this revolution.

David explains how it has changed the game of enterprise software: The old model was top-down, slow, inflexible and expensive. The new software development model is bottom-up, fast, flexible, and considerably less costly. A traditional proprietary stack is defined and controlled by a single vendor, or by a small group of vendors. It reflects the old command-and-control mentality of the traditional corporate world and the old economic order.

David then makes the case for an open-source analytics stack. For David, who is a leading proponent of open-source analytics, it's a logical leap: An open-source stack is defined by its community of users and contributors. No one "controls" an open-source stack, and no one can predict exactly how it will evolve. The open-source stack reflects the new realities of the networked global economy, which is increasingly dependent on big data.

It's certainly fair to argue whether the new analytics stack should be open, proprietary, or a blend of the two. From our perspective, it seems unlikely that large companies will abandon their investments in existing proprietary technologies overnight. Our hunch is that open-source and proprietary solutions will coexist for a long time, and for many good reasons. In fact, most proprietary vendors have been designing their solutions to plug and play with technology such as Hadoop. For example, Teradata Aster designed SQL-H, which is a seamless way to execute SQL and SQL-MapReduce on Apache Hadoop data.

CLOUD AND BIG DATA

It is important to remember that for all kinds of reasons—technical, political, social, regulatory, and cultural—cloud computing has not been a successful business model that has been widely adopted for enterprises to store their Big Data assets. However, there are many who believe that some obvious industry verticals will soon realize that there is a huge ROI opportunity if they do embrace the cloud.

Abhishek Mehta is one of those individuals who believes that cloud models are inevitable for every industry and it's just a matter of when an industry will shift to the cloud model. He explains that his clients are saying, "I don't have unlimited capital to invest in infrastructure.

My data is exploding—both structured and unstructured. The models that I use to price products or manage risks are broken. I'm under immense pressure to streamline my operations and reduce headcount. How am I going to solve these problems?" Market economics are demanding that capital-intensive infrastructure costs disappear and business challenges are forcing clients to consider newer models.

At the crossroads of high capital costs and rapidly changing business needs is a sea change that is driving the need for a new, compelling value proposition that is being manifested in a cloud-deployment model. With a cloud model, you pay on a subscription basis with no upfront capital expense. You don't incur the typical 30 percent maintenance fees—and all the updates on the platform are automatically available. The traditional cost of value chains is being completely disintermediated by platforms—massively scalable platforms where the marginal cost to deliver an incremental product or service is zero.

The ability to build massively scalable platforms—platforms where you have the option to keep adding new products and services for zero additional cost—is giving rise to business models that weren't possible before. Mehta calls it "the next industrial revolution, where the raw material is data and data factories replace manufacturing factories." He pointed out a few guiding principles that his firm stands by:

1. Stop saying "cloud." It's not about the fact that it is virtual, but the true value lies in delivering software, data, and/or analytics in an "as a service" model. Whether that is in a private hosted model or a publicly shared one does not matter. The delivery, pricing, and consumption model matters.

2. Acknowledge the business issues. There is no point to make light of matters around information privacy, security, access, and delivery. These issues are real, more often than not heavily regulated by multiple government agencies, and unless dealt with in a solution, will kill any platform sell.

3. Fix some core technical gaps. Everything from the ability to run analytics at scale in a virtual environment to ensuring information processing and analytics authenticity are issues that need solutions and have to be fixed.

MOBILE BUSINESS INTELLIGENCE

Analytics on mobile devices is what some refer to as putting BI in your pocket. Mobile drives straight to the heart of simplicity and ease of use that has been a major barrier to BI adoption since day one. Mobile devices are a great leveling field where making complicated actions easy is the name of the game.

For example, a young child can use an iPad but not a laptop. As a result, this will drive broad-based adoption as much for the ease of use as for the mobility these devices offer. This will have an immense impact on the business intelligence sector. We interviewed Dan Kerzner, SVP Mobile at MicroStrategy, a leading provider of business intelligence software.

He has been in the BI space for quite a while. People have been talking about mobile BI for quite some time, especially since the 1999 release of the good-old BlackBerry. However, it seems as though we have finally hit an inflection point. Ease of Mobile Application Deployment Another inflection point for the industry is the development and deployment of mobile applications. In the past, that was controlled by the relationship with the carrier. It used to be that if you wanted to push out a mobile application, the only way you could get that application on the phone for the most part was to go through the carriers.

That meant there were development environments that were sometimes very proprietary or you had to develop one set of applications for one carrier and another set of applications for a different one, maybe a third for the RIM BlackBerry environment. It didn't lend itself to very fast detonation because there was a real channel control now. Kerzner elaborated: One of the things that's happened recently is that with the advent of these app stores and the maturing of the browsers on the devices into something much more powerful, now as a software provider, you can go directly to the end user. I can go to a corporation and say I'm going to roll out a powerful global reporting application that's also going to do deal approvals and it's going to totally change a whole business process. I think something that was previously locked in a desk will now give people insights into the purchasing patterns, as well as the ability to take that action. I can roll out that whole application—I never have to talk to anybody but that customer because the devices that everybody's lugging around are really little computers and of course you can put any software you want on a little computer and that really wasn't the case historically in the mobile space.

Three elements that have impacted the viability of mobile BI:

1. Location—the GPS component and location . . . know where you are in time as well as the movement.
2. It's not just about pushing data; you can transact with your smart phone based on information you get.
3. Multimedia functionality allows the visualization pieces to really come into play.

Three challenges with mobile BI include:

1. Managing standards for rolling out these devices.
2. Managing security (always a big challenge).
3. Managing “bring your own device,” where you have devices both owned by the company and devices owned by the individual, both contributing to productivity.

CROWD SOURCING ANALYTICS

In October 2006, Netflix, an online DVD rental business, announced a contest to create a new predictive model for recommending movies based on past user ratings. The grand prize was \$1,000,000! While this may seem like a PR gimmick, it wasn't. Netflix already had an algorithm to solve the problem but thought there was an opportunity to realize additional model "lift," which would translate to huge top-line revenue. Netflix was an innovator in a space now being termed crowdsourcing.

Crowdsourcing is a recognition that you can't possibly always have the best and brightest internal people to solve all your big problems. By creating an open, competitive environment with clear rules and goals, Netflix realized their objective and, yes, they did create a lot of buzz about their organization in the process.

Crowdsourcing is a great way to capitalize on the resources that can build algorithms and predictive models. Let's face it, you can't "grow" a Ph.D. (or big brain) overnight. It takes years of learning and experience to get the knowledge to create algorithms and predictive models. So crowdsourcing is a way to capitalize on the limited resources that are available in the marketplace. It's often been said that competition brings out the best in us. We are all attracted to contests; our passion for competing seems hardwired into our souls.

Apparently, even predictive modelers find the siren song of competition irresistible. That's what a small Australian firm, Kaggle, has discovered—when given the chance, data scientists love to duke it out, just like everyone else. Kaggle describes itself as "an innovative solution for statistical/analytics outsourcing." That's a very formal way of saying that Kaggle manages competitions among the world's best data scientists. Here's how it works: Corporations, governments, and research laboratories are confronted with complex statistical challenges.

They describe the problems to Kaggle and provide data sets. Kaggle converts the problems and the data into contests that are posted on its web site. The contests feature cash prizes ranging in value from \$100 to \$3 million. Kaggle's clients range in size from tiny start-ups to multinational corporations such as Ford Motor Company and government agencies such as NASA. According to Anthony Goldbloom, Kaggle's founder and CEO, "The idea is that someone comes to us with a problem, we put it up on our website, and then people from all over the world can compete to see who can produce the best solution."

In essence, Kaggle has developed a remarkably effective global platform for crowdsourcing thorny analytic problems. What's especially attractive about Kaggle's approach is that it is truly a win-win scenario—contestants get access to real-world data (that has been carefully "anonymized" to eliminate privacy concerns) and prize sponsors reap the benefits of the contestants' creativity.

Crowdsourcing is a disruptive business model whose roots are in technology but is extending beyond technology to other areas. There are various types of crowdsourcing, such as crowd voting, crowd purchasing, wisdom of crowds, crowd funding, and contests. Take for example: ■ 99designs.com/ , which does crowdsourcing of graphic design ■ agentanything.com/ , which posts “missions” where agents vie for to run errands ■ 33needs.com/ , which allows people to contribute to charitable programs that make a social impact.

INTER AND TRANS FIREWALL ANALYTICS

Over the last 100 years, supply chains have evolved to connect multiple companies and enable them to collaborate to create enormous value to the end consumer via concepts such as CPFR, VMI, and so on.

Decision science is witnessing a similar trend as enterprises are beginning to collaborate on insights across the value chain. For instance, in the health care industry, rich consumer insights can be generated by collaborating on data and insights from the health insurance provider, pharmacy delivering the drugs, and the drug manufacturer. In fact, this is not necessarily limited to companies within the traditional demand-supply value chain.

For example, there are instances where a retailer and a social media company can come together to share insights on consumer behavior that will benefit both players. Some of the more progressive companies are taking this a step further and working on leveraging the large volumes of data outside the firewall such as social data, location data, and so forth. In other words, it will be not very long before internal data and insights from within the firewall is no longer a differentiator. We see this trend as the move from intra- to inter- and trans-firewall analytics. Yesterday companies were doing functional silo-based analytics. Today they are doing intra-firewall analytics with data within the firewall.

Tomorrow they will be collaborating on insights with other companies to do inter-firewall analytics as well as leveraging the public domain spaces to do trans-firewall analytics (see Figure 3.1). As Figure 3.2 depicts, setting up inter-firewall and trans-firewall analytics can add significant value. However it does present some challenges. First, as one moves outside the firewall, the information-to-noise ratio increases, putting additional requirements on analytical methods and technology requirements. Further, organizations are often limited by a fear of collaboration and an over reliance on proprietary information.

The fear of collaboration is mostly driven by competitive fears, data privacy concerns, and proprietary orientations that limit opportunities for cross-organizational learning and innovation. While it is clear that the transition to an inter- and trans-firewall paradigm is not easy, we feel it will continue to grow and at some point it will become a key weapon, available for decisions scientists to drive disruptive value and efficiencies.

Organizations will need to complement just intra-firewall insights with inter- and trans-firewall analytics

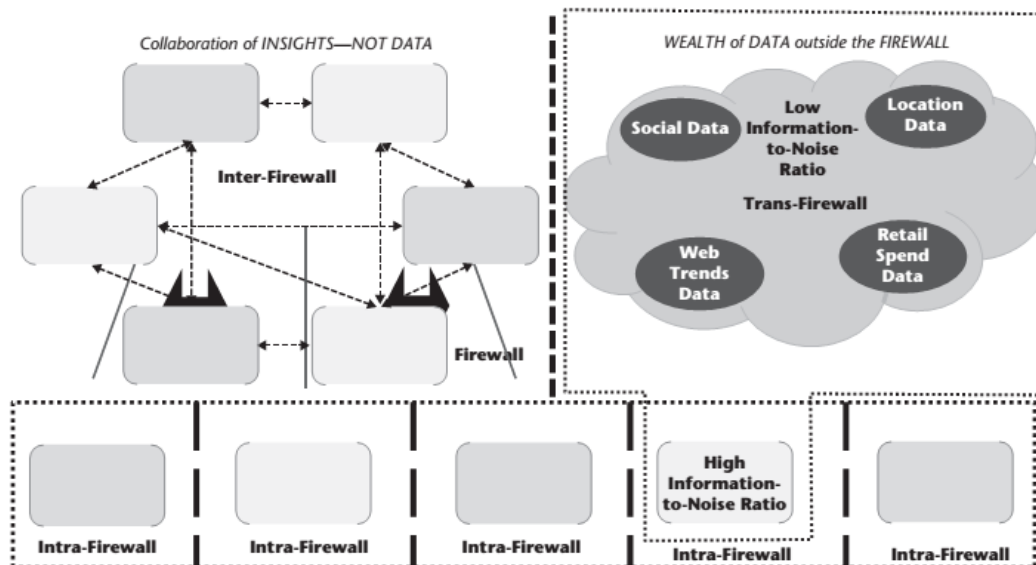


Figure 3.1 Inter- and Trans-Firewall Analytics
 Source: Mu Sigma and author Ambiga is a cofounder.

Disruptive value and efficiencies can be extracted by cooperating and exploring outside the boundaries of the firewall

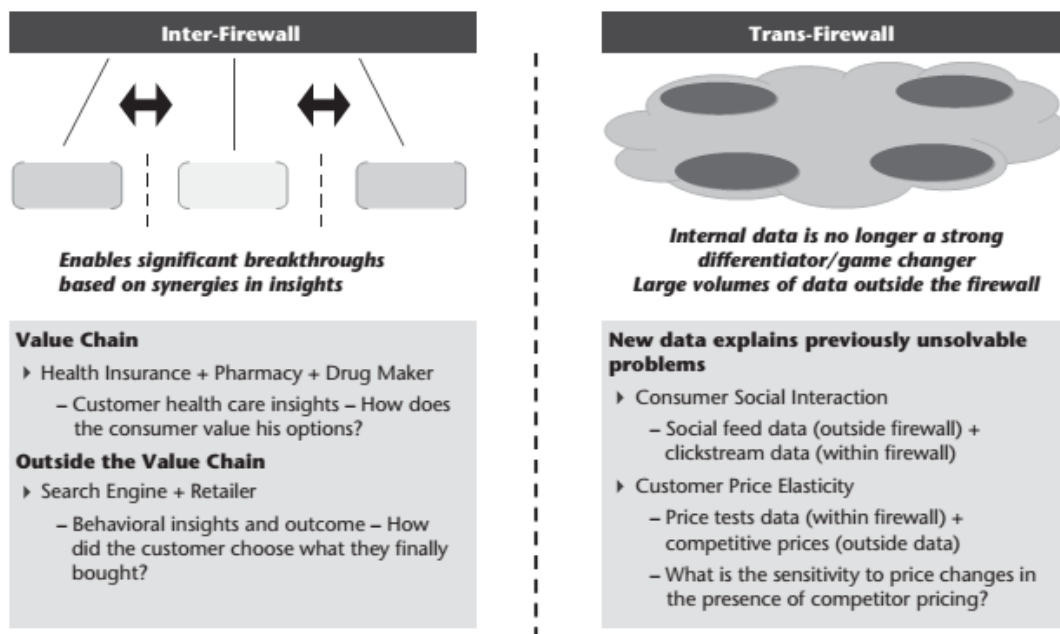


Figure 3.2 Value Chain for Inter-Firewall and Trans-Firewall Analytics
Source: Mu Sigma.

POSSIBLE QUESTIONS

PART-B **(6 MARKS)**

1. Explain about what is big data and why big data is needed?
2. Write a note on (i)big data technologies (ii)open source technologies
3. Write a note on (i)fraud and big data (ii)risk in big data
4. Describe the convergence of key trends in big data
5. Write a note on (i)fraud and big data (ii)risk in big data
6. Write a note on (i)Crowd sourcing analytics (ii) inter and trans firewall analytics

PART-C **(10 MARKS)**

1. Give a detail description about mobile business intelligence in big data.
2. Give a detail description about Graph databases Neo4J.
3. Give a brief explanation about big data technologies in detail.
4. Give a brief description about big data and healthcare in detail.
5. Briefly describe about big data and marketing in detail.



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS) (BATCH 2018-2020)

BIG DATA ANALYTICS

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

ONE MARK QUESTIONS

	UNIT-1	choice 1	choice 2	choice 3	choice 4			ANSWER
1	_____ is a Web search software	Imphala	Nutch	Oozie	Manmgy			Nutch
2	_____ defines an open application programming interface for common cloud application services.	Bigred	Nuvem	Oozie	Imphala			Nuvem
3	_____ is OData implementation in Java.	Bigred	Nuvem	Olingo	Onami			Onami
4	_____ is an open source SQL query engine for Apache HBase	Pig	Phoenix	Pivot	Manmgy			Phoenix
5	_____ provides multiple language implementations of the Advanced Messaged Queuing Protocol (AMQP)	RTA	Qpid	RAT	Nuvem			Qpid
6	_____ is A WEB And SOcial Mashup Engine.	ServiceMix	Samza	Rave	Oozie			Rave
7	The _____ project will create an ESB and component suite based on the Java Business Interface (JBI) standard.	ServiceMix	Samza	Rave	Pivot			Service Mix
8	_____ is spatial information system.	Sling	Solr	SIS	SLS			SIS
9	Stratos will be a polyglot _____ framework	Daas	PaaS	SaaS	RaaS			PaaS
10	_____ supports random-writable and advance-able sparse bitsets.	Stratos	Kafka	Sqoop	Lucene			Lucene

11	_____ is an open-source version control system.	Stratos	Kafka	Sqoop	Subversion			Subversion
12	_____ is a distributed data warehouse system for Hadoop.	Stratos	Tajo	Sqoop	Lucene			Tajo
13	_____ is a distributed, fault-tolerant, and high-performance realtime computation system.	Knife	Storm	Sqoop	Lucene			Storm
14	_____ is a standard compliant XML Query processor	Whirr	VXQuery	Knife	Lens			VXQuery
15	Apache _____ is a project that enables development and consumption of REST style web services.	Wives	Wink	Wig	Stratos			Wink
16	_____ is a log collection and correlation software with reporting and alarming functionalities.	Lucene	ALOIS	Imphal	Storm			ALOIS
17	_____ is a non-blocking, asynchronous, event driven high performance web framework.	AWS	AWF	AWT	ASW			AWF
18	_____ is the architectural center of Hadoop that allows multiple data processing engines.	YARN	Hive	Incubator	Chuckwa			YARN
19	YARN's dynamic allocation of cluster resources improves utilization over more static _____ rules used in early versions of Hadoop.	Hive	MapReduce	Imphala	Wink			MapReduce
20	The _____ is a framework-specific entity that negotiates resources from the Resource Manager.	NodeManager	Resource Manager	ApplicationMaster	TaskMaster			ApplicationMaster

21	Apache Hadoop YARN stands for :	Yet Another Reserve Negotiator	Yet Another Resource Network	Yet Another Resource Negotiator	Yet Any Resource Negotiator			Yet Another Resource Negotiator
22	MapReduce has undergone a complete overhaul in hadoop :	0.21	0.23	0.24	0.26			0.23
23	The _____ is the ultimate authority that arbitrates resources among all the applications in the system.	NodeManager	Resource Manager	ApplicationMaster	dynamic Master			ResourceManager
24	The _____ is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc.	Manager	Master	Scheduler	Task			Scheduler
25	The CapacityScheduler supports _____ queues to allow for more predictable sharing of cluster resources.	Networked	Hierarchical	Partition	Unpartition			Hierarchical
26	Zookeeper itself is intended to be replicated over a sets of hosts called _____.	chunks	ensemble	subdomains	unchunks			ensemble
27	_____ of the guarantee is provided by Zookeeper	Interactivity	Flexibility	Scalability	Reliability			Reliability
28	Zookeeper is especially fast in _____	write	read-dominant	read-write	read			read-dominant
29	_____ which a _____ is triggered the client receives a packet saying that the znode has _____.	event	watch	row	value			watch
30	The underlying client-server protocol has changed in version _____ of ZooKeeper.	2.0.0	3.0.0	4.0.0	6.0.0			3.0.0

31	A number of constants used in the client ZooKeeper API were renamed in order to	value	namespa ce	counter	signal			namesp ace
32	ZooKeeper allows distributed processes to coordinate with each other through registers,	znodes	hnodes	vnodes	rnodes			znodes
33	The HDFS client software implements _____ checking on the contents of HDFS	metastore	parity	checks um	unparity			checks um
34	The _____ machine is a single point of failure for an HDFS	DataNod e	NameNo de	ActionN ode	UnitNod e			NameN ode
35	The _____ and the EditLog are central data structures of HDFS.	DsImage	FsImage	FsImages	DsImage s			FsImag e
36	_____ support storing a copy of data at a particular instant of	Data Image	Datanots	Snapshot s	DataFile			Snapsh ots
37	Automatic restart and _____ of the NameNode software to another machine is not supported.	failover	end	scalabilit y	resource			failover
38	HDFS, by default, replicates each data block _____ times on different nodes and on at	3,2	1,2	2,3	3,1			3,2
39	_____ stores its metadata on multiple disks that typically include a non-local file	DataNod e	NameNo de	ActionN ode	UnitNod e			NameN ode
40	The HDFS file system is temporarily unavailable whenever the HDFS _____ is down.	DataNod e	NameNo de	ActionN ode	UnitNod e			NameN ode

41	_____ is a platform for constructing data flows for extract, transform, and load (ETL) processing and analysis of large datasets.	Pig Latin	Oozie	Pig	Hive			Pig
42	_____ hides the limitations of Java behind a powerful and concise Clojure API for Cascading.	Scalding	HCatalog	Cascalog	Catalog			Cascalog
43	Hive also support custom extensions written in :	C#	Java	C	C++			Java
44	_____ is the most popular high-level Java API in Hadoop	Scalding	HCatalog	Cascalog	Cascading			Cascading
45	_____ is general-purpose computing model and runtime system for distributed programming.	Mapreduce	Drill	Oozie	ActionNode			Mapreduce
46	The Pig Latin scripting language is not only a higher-level data flow language but also has operators similar to SQL.	SQL	JSON	XML	BSON			SQL
47	_____ jobs are optimized for scalability but not latency.	Mapreduce	Drill	Oozie	Hive			Hive
48	_____ is a framework for performing remote procedure calls and data serialization.	Drill	BigTop	Avro	Chukwa			Avro
49	Avro-backed tables can simply be created by using _____ in a DDL statement.	“STORED AS AVRO”	“STORED AS HIVE”	“STORED AS AVROHIVE”	“STORED AS SERDE”			“STORED AS AVRO”
50	Types that may be null must be defined as a _____ of that type and Null within Avro.	Union	Intersection	Set	Unset			Union
51	The files that are written by the _____ job are valid Avro files.	Avro	Map Reduce	Hive	Drill			Hive

52	Use _____ and embed the schema in the create statement.	schema.literal	schema.literal	row.literal	row.literal			schema.literal
53	_____ is interpolated into the quotes to correctly handle spaces within the schema.	\$\$SCHEMA	\$\$ROW	\$\$SCHEMASPACES	\$\$NAME SPACES			\$\$SCHEMA
54	_____ was designed to overcome limitations of the other Hive file formats.	ORC	OPC	ODC	OLC			ORC
55	An ORC file contains groups of row data called _____ :	postscript	stripes	script	scriptstart			stripes
56	The mapper implementation processes one line at a time.	map	reduce	mapper	reducer			map
57	The riiooop MapReduce framework spawns one map task for each _____ generated by the InputFormat for the job.	OutputSplit	InputSplit	InputSplitStream	OutputSplitStream			InputSplit
58	Users can control which keys (and hence records) go to which Reducer by implementing a custom _____ :	Partitioner	OutputSplit	Reporter	InputSplit			Partitioner
59	Applications can use the _____ to report progress and set application-level status.	Partitioner	OutputSplit	Reporter	InputSplit			Reporter
60	The right level of parallelism for maps seems to be around _____.	1-10	10-100	100-150	150-200			10-100

UNIT II

Syllabus

History of Hadoop- The Hadoop Distributed File System – Components of Hadoop- Analyzing the Data with Hadoop- Scaling Out- Hadoop Streaming- Design of HDFS-How Map Reduce Works-Anatomy of a Map Reduce Job run-Failures-Job Scheduling-Shuffle and Sort – Task execution - Map Reduce Types and Formats- Map Reduce Features

HISTORY OF HADOOP

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

The Origin of the Name “Hadoop”

The name Hadoop is not an acronym; it’s a made-up name. The project’s 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.

Projects in the Hadoop ecosystem also tend to have names that are unrelated to their function, often with an elephant or other animal theme (“Pig,” for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the namenode manages the filesystem namespace.

Building a web search engine from scratch was an ambitious goal, for not only is the software required to crawl and index websites complex to write, but it is also a challenge to run without a dedicated operations team, since there are so many moving parts. It’s expensive, too: Mike Cafarella and Doug Cutting estimated a system supporting a one-billion-page index would cost around \$500,000 in hardware, with a monthly running cost of \$30,000.

Nevertheless, they believed it was a worthy goal, as it would open up and ultimately democratize search engine algorithms.

Nutch was started in 2002, and a working crawler and search system quickly emerged. However, its creators realized that their architecture wouldn't scale to the billions of pages on the Web. Help was at hand with the publication of a paper in 2003 that described the architecture of Google's distributed filesystem, called GFS, which was being used in production at Google.

GFS, or something like it, would solve their storage needs for the very large files generated as a part of the web crawl and indexing process. In particular, GFS would free up time being spent on administrative tasks such as managing storage nodes. In 2004, Nutch's developers set about writing an open source implementation, the Nutch Distributed Filesystem (NDFS).

In 2004, Google published the paper that introduced MapReduce to the world. Early in 2005, the Nutch developers had a working MapReduce implementation in Nutch, and by the middle of that year all the major Nutch algorithms had been ported to run using MapReduce and NDFS.

NDFS and the MapReduce implementation in Nutch were applicable beyond the realm of search, and in February 2006 they moved out of Nutch to form an independent subproject of Lucene called Hadoop. At around the same time, Doug Cutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that ran at web scale (see the following sidebar). This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster.

Building Internet-scale search engines requires huge amounts of data and therefore large numbers of machines to process it. Yahoo! Search consists of four primary components: the Crawler, which downloads pages from web servers; the WebMap, which builds a graph of the known Web; the Indexer, which builds a reverse index to the best pages;

and the Runtime, which answers users' queries. The WebMap is a graph that consists of roughly 1 trillion (10^{12}) edges, each representing a web link, and 100 billion (10^{11}) nodes, each representing distinct URLs. Creating and analyzing such a large graph requires a large number of

computers running for many days. In early 2005, the infrastructure for the WebMap, named Dreadnaught, needed to be redesigned to scale up to more nodes.

Dreadnaught had successfully scaled from 20 to 600 nodes, but required a complete redesign to scale out further. Dreadnaught is similar to MapReduce in many ways, but provides more flexibility and less structure. In particular, each fragment in a Dreadnaught job could send output to each of the fragments in the next stage of the job, but the sort was all done in library code. In practice, most of the WebMap phases were pairs that corresponded to MapReduce. Therefore, the WebMap applications would not require extensive refactoring to fit into MapReduce.

Eric Baldeschwieler created a small team, and we started designing and Prototyping a new framework, written in C++ modeled and after GFS and MapReduce, to replace Dreadnaught. Although the immediate need was for a new framework for WebMap, it was clear that standardization of the batch platform across Yahoo! Search was critical and that by making the framework general enough to support other users, we could better leverage investment in the new platform.

At the same time, we were watching Hadoop, which was part of Nutch, and its progress.

In January 2006, Yahoo! hired Doug Cutting, and a month later we decided to abandon our prototype and adopt Hadoop. The advantage of Hadoop over our prototype and design was that it was already working with a real application (Nutch) on 20 nodes. That allowed us to bring up a research cluster two months later and start helping real customers use the new framework much sooner than we could have otherwise. Another advantage, of course, was that since Hadoop was already open source, it was easier (although far from easy!) to get permission from Yahoo!'s legal department to work in open source. So, we set up a 200-node cluster for the researchers in early 2006 and put the WebMap conversion plans on hold while we supported and improved Hadoop for the research users.—Owen O'Malley, 2009.

In January 2008, Hadoop was made its own top-level project at Apache, confirming its success and its diverse, active community. By this time,

Hadoop was being used by many other companies besides Yahoo!, such as Last.fm, Facebook, and the New York Times.

In one well-publicized feat, the New York Times used Amazon's EC2 compute cloud to crunch through 4 terabytes of scanned archives from the paper, converting them to PDFs for the Web.

The processing took less than 24 hours to run using 100 machines, and the project probably wouldn't have been embarked upon without the combination of Amazon's pay-by-the-hour model (which allowed the NYT to access a large number of machines for a short period) and Hadoop's easy-to-use parallel programming model.

In April 2008, Hadoop broke a world record to become the fastest system to sort an entire terabyte of data. Running on a 910-node cluster, Hadoop sorted 1 terabyte in 209 seconds (just under 3.5 minutes), beating the previous year's winner of 297 seconds.

In November of the same year, Google reported that its MapReduce implementation sorted 1 terabyte in 68 seconds.

Then, in April 2009, it was announced that a team at Yahoo! had used Hadoop to sort 1 terabyte in 62 seconds. The trend since then has been to sort even larger volumes of data at ever faster rates. In the 2014 competition, a team from Databricks were joint winners of the Gray Sort benchmark. They used a 207-node Spark cluster to sort 100 terabytes of data in 1,406 seconds, a rate of 4.27 terabytes per minute.

Today, Hadoop is widely used in mainstream enterprises. Hadoop's role as a generalpurpose storage and analysis platform for big data has been recognized by the industry, and this fact is reflected in the number of products that use or incorporate Hadoop in some way. Commercial Hadoop support is available from large, established enterprise vendors, including EMC, IBM, Microsoft, and Oracle, as well as from specialist Hadoop companies such as Cloudera, Hortonworks, and MapR.

THE HADOOP DISTRIBUTED FILE SYSTEM:

HDFS is an Apache Software Foundation project and a subproject of the Apache Hadoop project (see Related topics). Hadoop is ideal for

storing large amounts of data, like terabytes and petabytes, and uses HDFS as its storage system. HDFS lets you connect nodes (commodity personal computers) contained within clusters over which data files are distributed. You can then access and store the data files as one seamless file system. Access to data files is handled in a streaming manner, meaning that applications or commands are executed directly using the MapReduce processing model

HDFS is fault tolerant and provides high-throughput access to large data sets. This article explores the primary features of HDFS and provides a high-level view of the HDFS architecture.

HDFS has many similarities with other distributed file systems, but is different in several respects. One noticeable difference is HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high-throughput access. Another unique attribute of HDFS is the viewpoint that it is usually better to locate processing logic near the data rather than moving the data to the application space.

HDFS rigorously restricts data writing to one writer at a time. Bytes are always appended to the end of a stream, and byte streams are guaranteed to be stored in the order written.

HDFS has many goals. Here are some of the most notable:

Fault tolerance by detecting faults and applying quick, automatic recovery

Data access via MapReduce streaming

Simple and robust coherency model

ability across heterogeneous commodity hardware and operating systems

Scalability to reliably store and process large amounts of data

Economy by distributing data and processing across clusters of commodity personal computers

Efficiency by distributing data and logic to process it in parallel on nodes where data is located

Reliability by automatically maintaining multiple copies of data and automatically redeploying processing logic in the event of failures

HDFS provides interfaces for applications to move them closer to where the data is located, as described in the following section.

Application interfaces into HDFS

You can access HDFS in many different ways. HDFS provides a native Java™ application programming interface (API) and a native C-language wrapper for the Java API. In addition, you can use a web browser to browse HDFS files.

The applications described in Table 1 are also available to interface with HDFS.

Applications that can interface with HDFS:

Application	Description
-------------	-------------

FileSystem (FS) shell	
-----------------------	--

	A command-line interface similar to common Linux® and UNIX® shells (bash, csh, etc.) that allows interaction with HDFS data.
--	--

DFSAdmin	
----------	--

	A command set that you can use to administer an HDFS cluster.
--	---

fsck	
------	--

	A subcommand of the Hadoop command/application. You can use the fsck command to check for inconsistencies with files, such as missing blocks, but you cannot use the fsck command to correct these inconsistencies.
--	---

Name nodes and data nodes

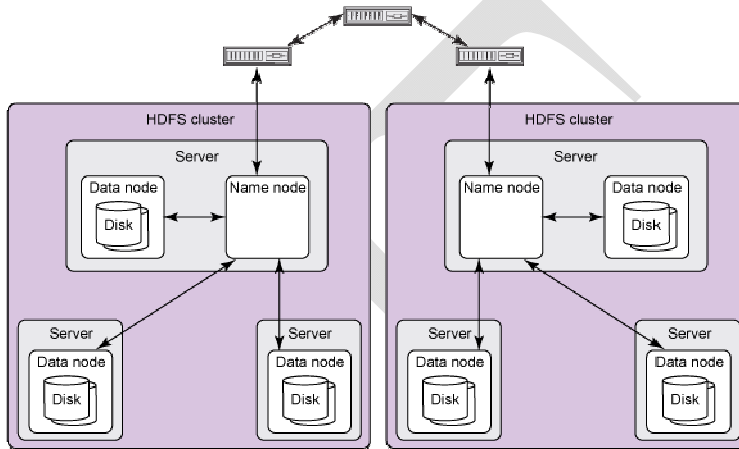
These have built-in web servers that let administrators check the current status of a cluster.

HDFS architecture

HDFS is comprised of interconnected clusters of nodes where files and directories reside. An HDFS cluster consists of a single node, known as a NameNode, that manages the file system namespace and regulates client access to files. In addition, data nodes (DataNodes) store data as blocks within files.

Name nodes and data nodes

Within HDFS, a given name node manages file system namespace operations like opening, closing, and renaming files and directories. A name node also maps data blocks to data nodes, which handle read and write requests from HDFS clients. Data nodes also create, delete, and replicate data blocks according to instructions from the governing name node.



As Figure 1 illustrates, each cluster contains one name node. This design facilitates a simplified model for managing each namespace and arbitrating data distribution.

Relationships between name nodes and data nodes

Name nodes and data nodes are software components designed to run in a decoupled manner on commodity machines across heterogeneous operating systems. HDFS is built using the Java programming language; therefore, any machine that supports the Java programming language can run HDFS. A typical installation cluster has a dedicated machine that runs a name node and possibly one data node. Each of the other machines in the cluster runs one data node.

As Figure 1 illustrates, each cluster contains one name node. This design facilitates a simplified model for managing each namespace and arbitrating data distribution.

Relationships between name nodes and data nodes

Name nodes and data nodes are software components designed to run in a decoupled manner on commodity machines across heterogeneous operating systems. HDFS is built using the Java programming language; therefore, any machine that supports the Java programming language can run HDFS. A typical installation cluster has a dedicated machine that runs a name node and possibly one data node. Each of the other machines in the cluster runs one data node.

The name node maintains and administers changes to the file system namespace.

File system namespace

HDFS supports a traditional hierarchical file organization in which a user or an application can create directories and store files inside them. The file system namespace hierarchy is similar to most other existing file systems; you can create, rename, relocate, and remove files.

COMPONENTS OF A HADOOP:

The Hadoop Ecosystem comprises of 4 core components –

1) Hadoop Common-

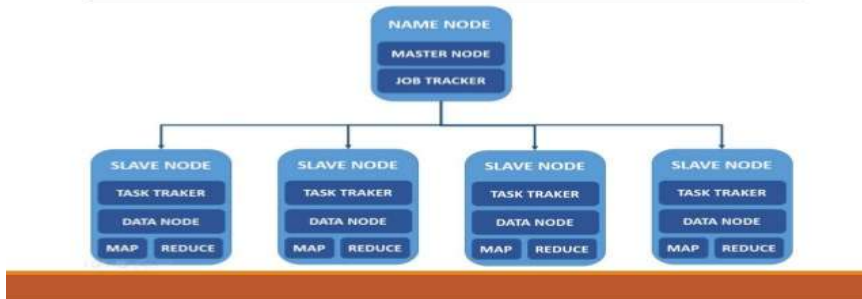
Apache Foundation has pre-defined set of utilities and libraries that can be used by other modules within the Hadoop ecosystem. For example, if HBase and Hive want to access HDFS they need to make of Java archives (JAR files) that are stored in Hadoop Common.

2)) Hadoop Distributed File System (HDFS) -

The default big data storage layer for Apache Hadoop is HDFS. HDFS is the “Secret Sauce” of Apache Hadoop components as users can dump huge datasets into HDFS and the data will sit there nicely until the user wants to leverage it for analysis. HDFS component creates several replicas of the data block to be distributed across different clusters for reliable and quick data access. HDFS comprises of 3 important components-NameNode, DataNode and Secondary NameNode. HDFS operates on a Master-Slave architecture model where the NameNode acts as the master node for keeping a track of the storage cluster and the DataNode acts as a slave node summing up to the various systems within a Hadoop cluster.

HDFS Use Case-

HADOOP MASTER/SLAVE ARCHITECTURE



Nokia deals with more than 500 terabytes of unstructured data and close to 100 terabytes of structured data. Nokia uses HDFS for storing all the structured and unstructured data sets as it allows processing of the stored data at a petabyte scale.

3)) MapReduce- Distributed Data Processing Framework of Apache Hadoop

MapReduce is a Java-based system created by Google where the actual data from the HDFS store gets processed efficiently. MapReduce breaks down a big data processing job into smaller tasks. MapReduce is responsible for the analysing large datasets in parallel before reducing it to find the results. In the Hadoop ecosystem, Hadoop MapReduce is a framework based on YARN architecture.

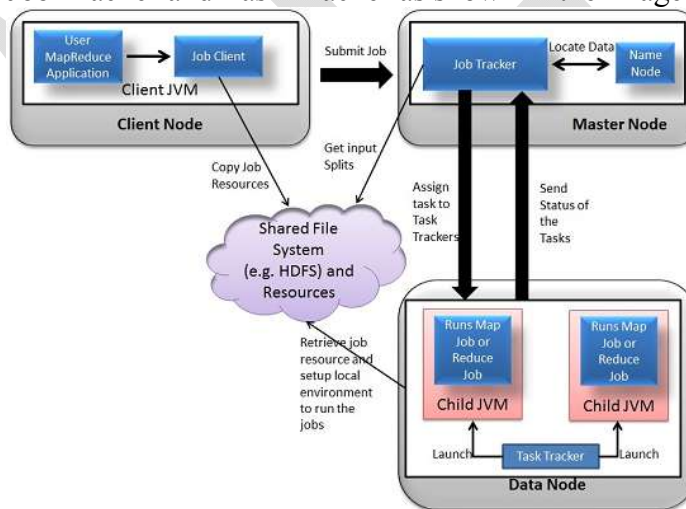
YARN based Hadoop architecture, supports parallel processing of huge data sets and MapReduce provides the framework for easily writing applications on thousands of nodes, considering fault and failure management.

The basic principle of operation behind MapReduce is that the “Map” job sends a query for processing to various nodes in a Hadoop cluster and the “Reduce” job collects all the results to output into a single value. Map Task in the Hadoop ecosystem takes input data and splits into independent

chunks and output of this task will be the input for Reduce Task. In The same Hadoop ecosystem Reduce task combines Mapped data tuples into smaller set of tuples. Meanwhile, both input and output of tasks are stored in a file system. MapReduce takes care of scheduling jobs, monitoring jobs and re-executes the failed task.

MapReduce framework forms the compute node while the HDFS file system forms the data node. Typically in the Hadoop ecosystem architecture both data node and compute node are considered to be the same.

The delegation tasks of the MapReduce component are tackled by two daemons- Job Tracker and Task Tracker as shown in the image below –



4)YARN

YARN forms an integral part of Hadoop 2.0.YARN is great enabler for dynamic resource utilization on Hadoop framework as users can run various Hadoop applications without having to bother about increasing workloads.

Hadoop 1.0 vs Hadoop 2.0 YARN

Key Benefits of Hadoop 2.0 YARN Component-

Hadoop 2 - YARN Architecture

ResourceManager (RM)

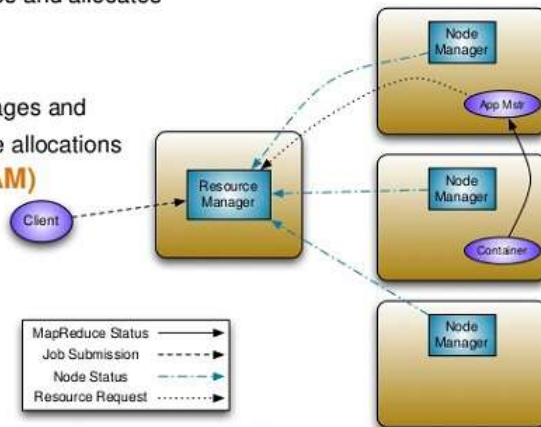
Central agent - Manages and allocates
cluster resources

NodeManager (NM)

Per-Node agent - Manages and
enforces node resource allocations

ApplicationMaster (AM)

Per-Application –
Manages application –
lifecycle and task
scheduling



It offers improved cluster utilization

Highly scalable

Beyond Java

Novel programming models and services

Agility

YARN Use Case:

Yahoo has close to 40,000 nodes running Apache Hadoop with 500,000 MapReduce jobs per day taking 230 compute years extra for processing every day. YARN at Yahoo helped them increase the load on the most heavily used Hadoop cluster to 125,000 jobs a day when compared to 80,000 jobs a day which is close to 50% increase.

ANALYZING THE DATA WITH HADOOP

With rapid innovations, frequent evolutions of technologies and a rapidly growing internet population, systems and enterprises are generating huge amounts of data to the tune of terabytes and even petabytes of information. Since data is being generated in very huge volumes with great velocity in all multi-structured formats like images, videos, weblogs, sensor data, etc. from all different sources, there is a huge demand to efficiently store, process and analyze this large amount of data to make it usable.

Hadoop is undoubtedly the preferred choice for such a requirement due to its key characteristics of being reliable, flexible, economical, and a scalable solution. While Hadoop provides the ability to store this large scale data on HDFS (Hadoop Distributed File System), there are multiple solutions available in the market for analyzing this huge data like MapReduce, Pig and Hive. With the advancements of these different data analysis technologies to analyze the big data, there are many different school of thoughts about which Hadoop data analysis technology should be used when and which could be efficient.

A well-executed big data analysis provides the possibility to uncover hidden markets, discover unfulfilled customer demands and cost reduction opportunities and drive game-changing, significant improvements in everything from telecommunication efficiencies and surgical or medical treatments, to social media campaigns and related digital marketing promotions.

HDFS sends data to the server once and uses it as many times as it wants. When a query is raised, NameNode manages all the DataNode slave nodes that serve the given query. Hadoop MapReduce performs all the jobs assigned sequentially. Instead of MapReduce, Pig Hadoop and Hive Hadoop are used for better performances.

Other packages that can support Hadoop are listed below.

Apache Oozie: A scheduling system that manages processes taking place in Hadoop

Apache Pig: A platform to run programs made on Hadoop

Cloudera Impala: A processing database for Hadoop. Originally it was created by the software organisation Cloudera, but was later released as open source software

Apache HBase: A non-relational database for Hadoop

Apache Phoenix: A relational database based on Apache HBase

Apache Hive: A data warehouse used for summarisation, querying and the analysis of data

Apache Sqoop: Is used to store data between Hadoop and structured data sources

Apache Flume: A tool used to move data to HDFS

Cassandra: A scalable multi-database system

The importance of Hadoop

Hadoop is capable of storing and processing large amounts of data of various kinds. There is no need to preprocess the data before storing it. Hadoop is highly scalable as it can store and distribute large data sets over several machines running in parallel. This framework is free and uses cost-efficient methods.

Hadoop is used for:

Machine learning

Processing of text documents
Image processing
Processing of XML messages
Web crawling
Data analysis
Analysis in the marketing field
Study of statistical data

Challenges when using Hadoop

Hadoop does not provide easy tools for removing noise from the data; hence, maintaining that data is a challenge. It has many data security issues like encryption problems. Streaming jobs and batch jobs are not performed efficiently. MapReduce programming is inefficient for jobs involving highly analytical skills. It is a distributed system with low level APIs. Some APIs are not useful to developers.

But there are benefits too. Hadoop has many useful functions like data warehousing, fraud detection and marketing campaign analysis. These are helpful to get useful information from the collected data. Hadoop has the ability to duplicate data automatically. So multiple copies of data are used as a backup to prevent loss of data.

Frameworks similar to Hadoop

Any discussion on Big Data is never complete without a mention of Hadoop. But like with other technologies, a variety of frameworks that are similar to Hadoop have been developed. Other frameworks used widely are Ceph, Apache Storm, Apache Spark, DataTorrentRTS, Google BiqQuery, Samza, Flink and HydraDataTorrentRTS.

MapReduce requires a lot of time to perform assigned tasks. Spark can fix this issue by doing in-memory processing of data. Flink is another framework that works faster than Hadoop and Spark. Hadoop is not efficient for real-time processing of data. Apache Spark uses stream processing of data where continuous input and output of data happens. Apache Flink also provides single runtime for the streaming of data and batch processing.

However, Hadoop is the preferred platform for Big Data analytics because of its scalability, low cost and flexibility. It offers an array of tools that data scientists need. Apache Hadoop with YARN transforms a large set of raw data into a feature matrix which is easily consumed. Hadoop makes machine learning algorithms easier.

Scaling Out

To scale out, we need to store the data in a distributed filesystem. This allows Hadoop to move the MapReduce computation to each machine hosting a part of the data, using Hadoop's resource management system, called YARN.

Data

Flow

First, some terminology. A MapReduce *job* is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information. Hadoop runs the job by dividing it into *tasks*, of which there are two types: *map tasks* and *reduce tasks*. The tasks are scheduled using YARN and run on nodes in the cluster. If a task fails, it will be automatically rescheduled to run on a different node. Hadoop divides the input to a MapReduce job into fixed-size pieces called *input splits*, or just *splits*. Hadoop creates one map task for each split, which runs the user-defined map function for each *record* in the split. Having many splits means the time taken to process each split is small compared to the time to process the whole input. So if we are processing the splits in parallel, the processing is better load balanced when the splits are small, since a faster machine will be able to process proportionally more splits over the course of the job than a slower machine. Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing

increases as the splits become more fine grained.

On the other hand, if splits are too small, the overhead of managing the splits and map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of an HDFS block, which is 128 MB by default, although this can be changed for the cluster (for all newly created files) or specified when each file is created. Hadoop does its best to run the map task on a node where the input data resides in HDFS, because it doesn't use valuable cluster bandwidth. This is called the *data locality optimization*. Sometimes, however, all the nodes hosting the HDFS block replicas for a map task's input split are running other map tasks, so the job scheduler will look for a free map slot on a node in the same rack as one of the blocks. Very occasionally even this is not possible, so an off-rack node is used, which results in an inter-rack network transfer. The three possibilities are illustrated in **Figure 2-2**.

It should now be clear why the optimal split size is the same as the block size: it is the largest size of input that can be guaranteed to be stored on a single node. If the split spanned two blocks, it would be unlikely that any HDFS node stored both blocks, so some of the split would have to be transferred across the network to the node running the map task, which is clearly less efficient than running the whole map task using local data.

Map tasks write their output to the local disk, not to HDFS. Why is this? Map output is intermediate output: it's processed by reduce tasks to produce the final output, and once the job is complete, the map output can be thrown away. So, storing it in HDFS with replication would be overkill. If the node running the map task fails before the map output has been consumed by the reduce task, then Hadoop will automatically rerun the map task on another node to re-create the map output.

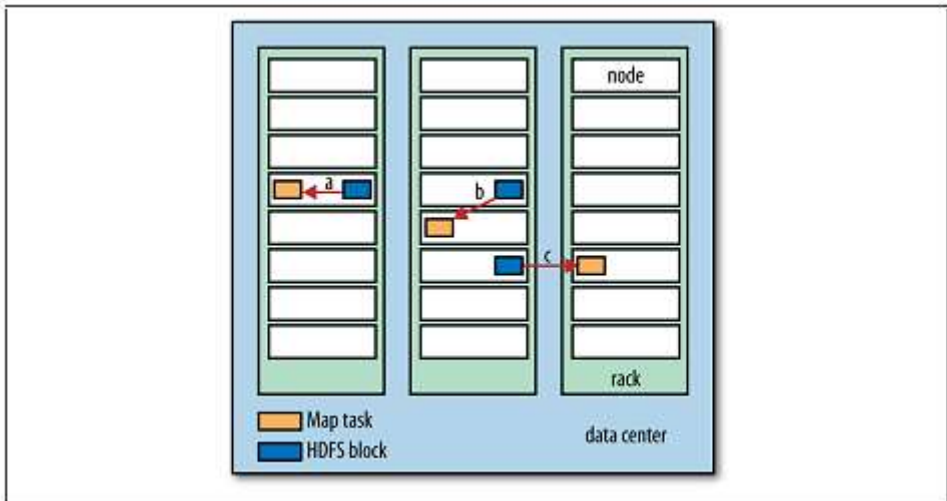


Figure 2-2. Data-local (a), rack-local (b), and off-rack (c) map tasks

Reduce tasks don't have the advantage of data locality; the input to a single reduce task is normally the output from all mappers. In the present example, we have a single reduce task that is fed by all of the map tasks. Therefore, the sorted map outputs have to be transferred across the network to the node where the reduce task is running, where they are merged and then passed to the user-defined reduce function. The output of the reduce is normally stored in HDFS for reliability. HDFS block of the reduce output, the first replica is stored on the local node, with other replicas being stored on off-rack nodes for reliability. Thus, writing the reduce output does consume network bandwidth, but only as much as a normal HDFS write pipeline consumes. The whole data flow with a single reduce task is illustrated in **Figure 2-3**. The dotted boxes indicate nodes, the dotted arrows show data transfers on a node, and the solid arrows show data transfers between nodes.

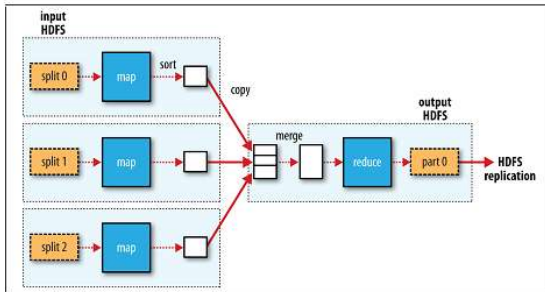


Figure 2-3. MapReduce data flow with a single reduce task

The number of reduce tasks is not governed by the size of the input, but instead is specified independently. When there are multiple reducers, the map tasks *partition* their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for any given key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well. The data flow for the general case of multiple reduce tasks. This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as “the shuffle,” as each reduce task is fed by many map tasks. The shuffle is more complicated than this diagram suggests, and tuning it can have a big impact on job execution time, as you will see in “**Shuffle and Sort**”

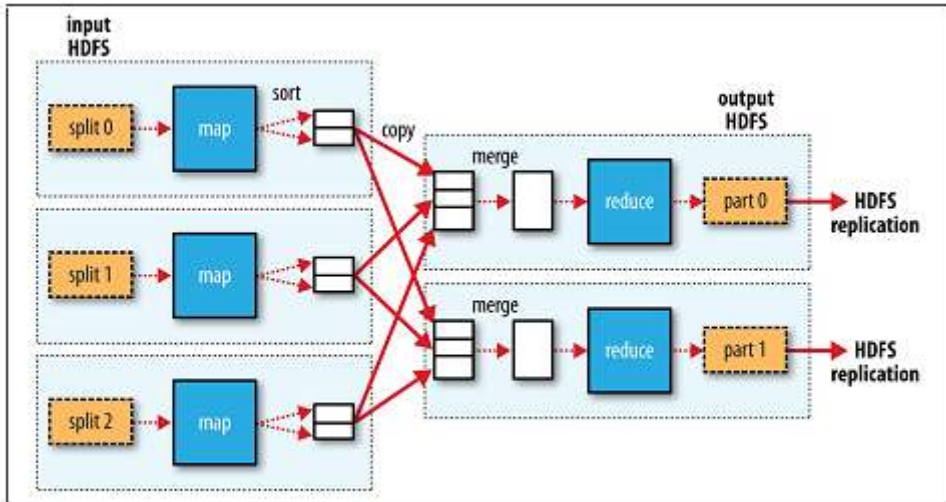


Figure 2-4. MapReduce data flow with multiple reduce tasks

Finally, it's also possible to have zero reduce tasks. This can be appropriate when you don't need the shuffle because the processing can be carried out entirely in parallel. In this case, the only off-node data transfer is when the map tasks write to HDFS Combiner Functions. Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a *combiner function* to be run on the map output, and the combiner function's output forms the input to the reduce function. Because the combiner function is an optimization, Hadoop does not provide a guarantee of how many times it will call it for a particular map output record, if at all. In other words, calling the combiner function zero, one, or many times should produce the same output from the reducer.

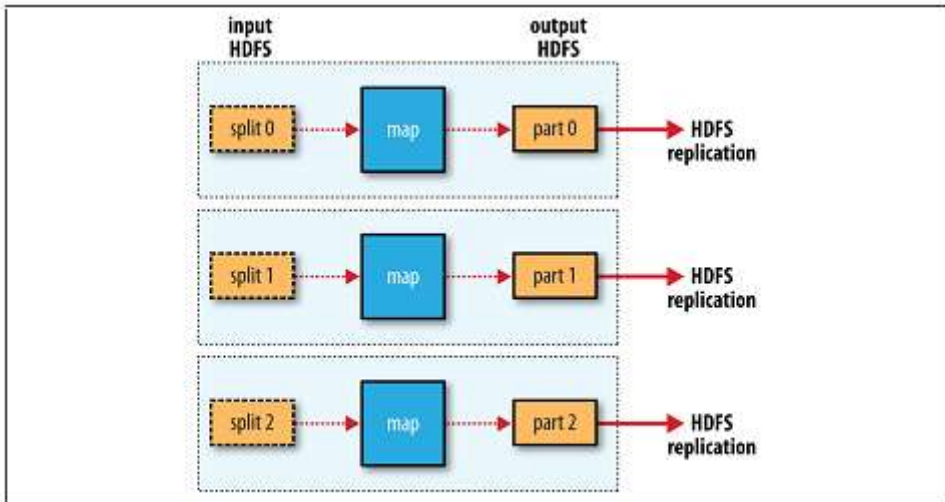


Figure 2-5. MapReduce data flow with no reduce tasks

The contract for the combiner function constrains the type of function that may be used.

This is best illustrated with an example. Suppose that for the maximum temperature example, readings for the year 1950 were processed by two maps (because they were in different splits). Imagine the first map produced the output:

(1950, 0)

(1950, 20)

(1950, 10)

and the second produced:

(1950, 25)

(1950, 15)

The reduce function would be called with a list of all the values:

(1950, [0, 20, 10, 25, 15])

with output:

(1950, 25)

since 25 is the maximum value in the list.

We could use a combiner function that, just like the reduce function, finds the maximum temperature for each map output. The reduce function would then be called with:

(1950, [20, 25])

and would produce the same output as before. More succinctly, we may express the

function calls on the temperature values in this case as follows:

$max(0, 20, 10, 25, 15) = max(max(0, 20, 10), max(25, 15)) = max(20, 25) = 25$

Not all functions possess this property.¹ For example, if we were calculating mean temperatures, we couldn't use the mean as our combiner function, because:

$mean(0, 20, 10, 25, 15) = 14$

but:

$mean(mean(0, 20, 10), mean(25, 15)) = mean(10, 20) = 15$

The combiner function doesn't replace the reduce function. (How could it?

The reduce function is still needed to process records with the same key from different maps.) But it can help cut down the amount of data shuffled between the mappers and the reducers, and for this reason alone it is always worth considering whether you can use a combiner function in your MapReduce job.

Specifying a combiner function

Going back to the Java MapReduce program, the combiner function is defined using the Reducer class, and for this application, it is the same implementation as the reduce function in MaxTemperatureReducer. The only change we need to make is to set the combiner class on the Job.

HADOOP STREAMING

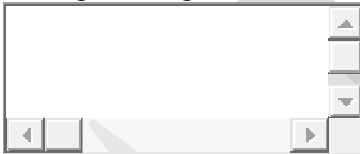
Streaming is naturally suited for text processing (although, as of version 0.21.0, it can handle binary streams, too), and when used in text mode, it has a line-oriented view of data. Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output. A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format a tab-separated key-value pair passed over standard

input. The reduce function reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

Python

Streaming supports any programming language that can read from standard input, and write to standard output, so for readers more familiar with Python, here's the same example again. The map script is in Example and the reduce script is in Example

Example . Map function for maximum temperature in Python



```
#!/usr/bin/env python
import re
import sys
for line in sys.stdin:
    val = line.strip()
    (year, temp, q) = (val[15:19], val[87:92], val[92:93])
    if (temp != "+9999" and re.match("[01459]", q)):
        print "%st%s" % (year, temp)
```

Example 2-11. Reduce function for maximum temperature in Python

```
#!/usr/bin/env python
import sys
(last_key, max_val) = (None, 0)
for line in sys.stdin:
    (key, val) = line.strip().split("t")
    if last_key and last_key != key:
        print "%st%s" % (last_key, max_val)
        (last_key, max_val) = (key, int(val))
    else:
        (last_key, max_val) = (key, max(max_val, int(val)))
    if last_key:
        print "%st%s" % (last_key, max_val)
```

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. *Hadoop Streaming* uses Unix standard streams as the interface between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce program. Streaming is naturally suited for text processing. Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output.

A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format—a tab-separated key-value pair passed over standard input. The reduce function reads lines from standard input, which the frame-work guarantees are sorted by key, and writes its results to standard output. Let's illustrate this by rewriting our MapReduce program for finding maximum temperatures by year in Streaming.

Ruby

The map function can be expressed in Ruby as shown in **Example 2-7**.

Example 2-7. Map function for maximum temperature in Ruby

```
#!/usr/bin/env ruby
STDIN.each_line do |line|
  val = line
  year, temp, q = val[15,4], val[87,5], val[92,1]
  puts "#{year}\t#{temp}" if (temp != "+9999" && q =~ /[01459]/)
end
```

The program iterates over lines from standard input by executing a block for each line from STDIN (a global constant of type IO). The block pulls out the relevant fields from each input line and, if the temperature is valid, writes the year and the temperature separated by a tab character, \t, to standard output (using puts).

Because the script just operates on standard input and output, it's trivial to test the script without using Hadoop, simply by using Unix pipes:

```
% cat input/ncdc/sample.txt | ch02-mr-  
intro/src/main/ruby/max_temperature_map.rb
```

1950 +0000
1950 +0022
1950 -0011
1949 +0111
1949 +0078

The reduce function shown in **Example 2-8** is a little more complex.

Example 2-8. Reduce function for maximum temperature in Ruby

```
#!/usr/bin/env ruby
last_key, max_val = nil, -1000000
STDIN.each_line do |line|
  key, val = line.split("\t")
  if last_key && last_key != key
    puts "#{last_key}\t#{max_val}"
    last_key, max_val = key, val.to_i
  else
    last_key, max_val = key, [max_val, val.to_i].max
  end
end
puts "#{last_key}\t#{max_val}" if last_key
```

Again, the program iterates over lines from standard input, but this time we have to store some state as we process each key group. In this case, the keys are the years, and we store the last key seen and the maximum temperature seen so far for that key. The MapReduce framework ensures that the keys are ordered, so we know that if a key is different from the previous one, we have moved into a new key group. In contrast to the Java API, where you are provided an iterator over each key group, in Streaming you have to find key group boundaries in your program. For each line, we pull out the key and value. Then, if we've just finished a group (`last_key && last_key != key`), we write the key and the maximum temperature for that group, separated by a tab character, before resetting the maximum temperature for the new key. If we haven't just finished a group, we just update the maximum temperature for the current key.

The last line of the program ensures that a line is written for the last key group in the input.

We can now simulate the whole MapReduce pipeline with a Unix pipeline

(which is equivalent to the Unix pipeline shown in **Figure 2-1**): % **cat
input/ncdc/sample.txt | \
ch02-mr-intro/src/main/ruby/max_temperature_map.rb | \
sort | ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb
1949 111
1950 22**

The output is the same as that of the Java program, so the next step is to run it using Hadoop itself.

The hadoop command doesn't support a Streaming option; instead, you specify the Streaming JAR file along with the jar option. Options to the Streaming program specify the input and output paths and the map and reduce scripts.

This is what it looks like:

```
% hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-  
streaming-*.jar \  
-input input/ncdc/sample.txt \  
-output output \  
-mapper ch02-mr-intro/src/main/ruby/max_temperature_map.rb \  
-reducer ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb
```

When running on a large dataset on a cluster, we should use the -combiner option to

set the combiner:

```
% hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-  
streaming-*.jar \  
-files ch02-mr-intro/src/main/ruby/max_temperature_map.rb,\  
ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb \  
-input input/ncdc/all \  
-output output \  
-mapper ch02-mr-intro/src/main/ruby/max_temperature_map.rb \  
-combiner ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb  
\  
-reducer ch02-mr-intro/src/main/ruby/max_temperature_reduce.rb
```

Note also the use of -files, which we use when running Streaming programs on the cluster to ship the scripts to the cluster.

Python

Streaming supports any programming language that can read from standard input and write to standard output, so for readers more familiar

with Python, here's the same example again.⁵ The map script is in Example 2-9, and the reduce script is in Example 2-10.

Example 2-9. Map function for maximum temperature in Python

```
#!/usr/bin/env python
```

```
import re
import sys
for line in sys.stdin:
    val = line.strip()
    (year, temp, q) = (val[15:19], val[87:92], val[92:93])
    if (temp != "+9999" and re.match("[01459]", q)):
        print "%s\t%s" % (year, temp)
```

Example 2-10. Reduce function for maximum temperature in Python

```
#!/usr/bin/env python
```

```
import sys
(last_key, max_val) = (None, -sys.maxint)
for line in sys.stdin:
    (key, val) = line.strip().split("\t")
    if last_key and last_key != key:
        print "%s\t%s" % (last_key, max_val)
        (last_key, max_val) = (key, int(val))
    else:
        (last_key, max_val) = (key, max(max_val, int(val)))
    if last_key:
        print "%s\t%s" % (last_key, max_val)
```

We can test the programs and run the job in the same way we did in Ruby.

For example,

to run a test:

```
% cat input/ncdc/sample.txt | \
ch02-mr-intro/src/main/python/max_temperature_map.py | \
sort | ch02-mr-intro/src/main/python/max_temperature_reduce.py
1949 111
1950 22
```

Design of HDFS

HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.

“Very large” in this context means files that are hundreds of megabytes, gigabytes, or terabytes in size. There are Hadoop clusters running today that store petabytes of data.²

Streaming data access

HDFS is built around the idea that the most efficient data processing pattern is a write-once, read-many-times pattern. A dataset is typically generated or copied from source, and then various analyses are performed on that dataset over time. Each analysis will involve a large proportion, if not all, of the dataset, so the time to read the whole dataset is more important than the latency in reading the first record.

Commodity hardware

Hadoop doesn’t require expensive, highly reliable hardware. It’s designed to run on clusters of commodity hardware (commonly available hardware that can be obtained from multiple vendors)³ for which the chance of node failure across the cluster is high, at least for large clusters. HDFS is designed to carry on working without a noticeable interruption to the user in the face of such failure. It is also worth examining the applications for which using HDFS does not work so well.

Although this may change in the future, these are areas where HDFS is not a good fit today:

Low-latency data access

Applications that require low-latency access to data, in the tens of milliseconds range, will not work well with HDFS. Remember, HDFS is optimized for delivering a high throughput of data, and this may be at the expense of latency. HBase is currently a better choice for low-latency access.

Lots of small files Because the namenode holds filesystem metadata in

memory, the limit to the number of files in a filesystem is governed by the amount of memory on the namenode. As a rule of thumb, each file, directory, and block takes about 150 bytes. So, for example, if you had one million files, each taking one block, you would need at least 300 MB of memory.

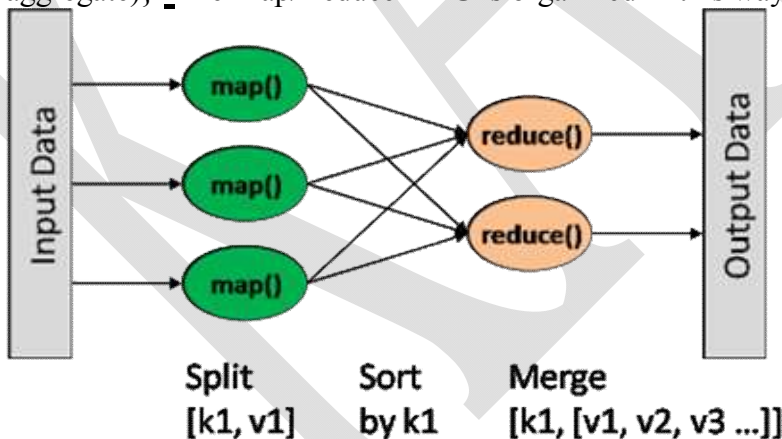
Although storing millions of files is feasible, billions is beyond the capability of current hardware. *Multiple writers, arbitrary file modifications* Files in HDFS may be written to by a single writer. Writes are always made at the end of the file, in append-only fashion. There is no support for multiple writers or for modifications at arbitrary offsets in the file. (These might be supported in the future, but they are likely to be relatively inefficient.)

How Map Reduce Works

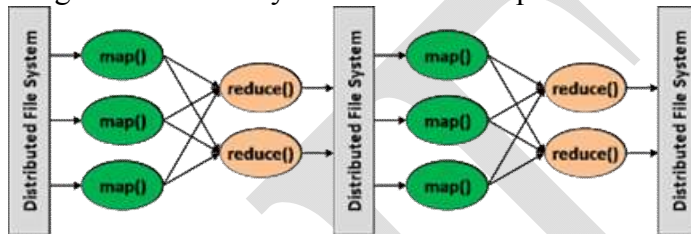
Map/reduce is a special form of such a DAG which is applicable in a wide range of use cases. It is organized as a “map” function which transform a piece of data into some number of key/value pairs. Each of these elements will then be sorted by their key and reach to the same node, where a “reduce” function is use to merge the values (of the same key) into a single result.

```
map(input_record) { ... emit(k1, v1) ... emit(k2, v2) ... }
reduce(key, values) { aggregate = initialize() while (values.has_next) { aggregate = merge(values.next) } collect(key, aggregate) }
```

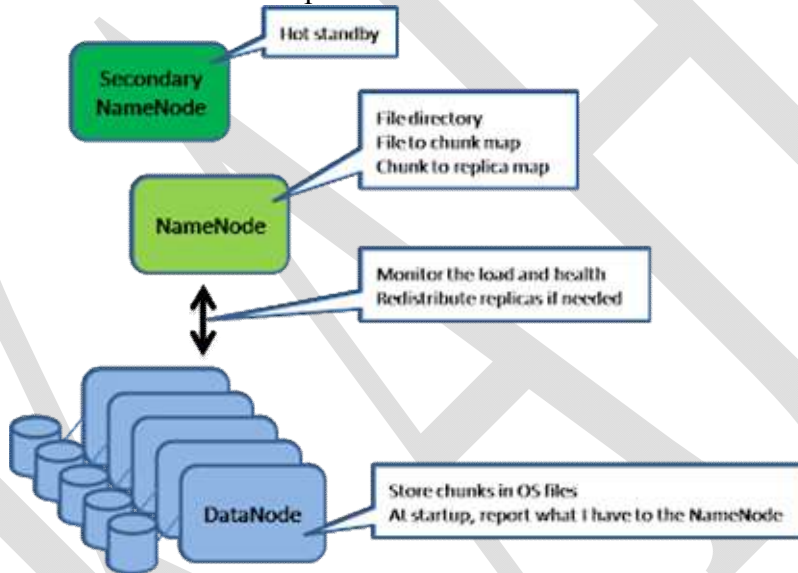
The Map/Reduce DAG is organized in this way.



A parallel algorithm is usually structure as multiple rounds of Map/Reduce



HDFS The distributed file system is designed to handle large files (multi-GB) with sequential read/write operation. Each file is broken into chunks, and stored across multiple data nodes as local OS files.



There is a master “NameNode” to keep track of overall file directory structure and the placement of chunks. This NameNode is the central control point and may re-distributed replicas as needed. DataNode reports all its chunks to the NameNode at bootup. Each chunk has a version number which will be increased for all update. Therefore, the NameNode know if any of the chunks of a DataNode is stale (e.g. when the DataNode

crash for some period of time). Those stale chunks will be garbage collected at a later time. To read a file, the client API will calculate the chunk index based on the offset of the file pointer and make a request to the NameNode. The NameNode will reply which DataNodes has a copy of that chunk. From this points, the client contacts the DataNode directly without going through the NameNode.

To write a file, client API will first contact the NameNode who will designate one of the replica as the primary (by granting it a lease). The response of the NameNode contains who is the primary and who are the secondary replicas. Then the client push its changes to all DataNodes in any order, but this change is stored in a buffer of each DataNode. After changes are buffered at all DataNodes, the client send a “commit” request to the primary, which determines an order to update and then push this order to all other secondaries. After all secondaries complete the commit, the primary will response to the client about the success. All changes of chunk distribution and metadata changes will be written to an operation log file at the NameNode. This log file maintain an order list of operation which is important for the NameNode to recover its view after a crash. The NameNode also maintain its persistent state by regularly check-pointing to a file. In case of the NameNode crash, a new NameNode will take over after restoring the state from the last checkpoint file and replay the operation log.

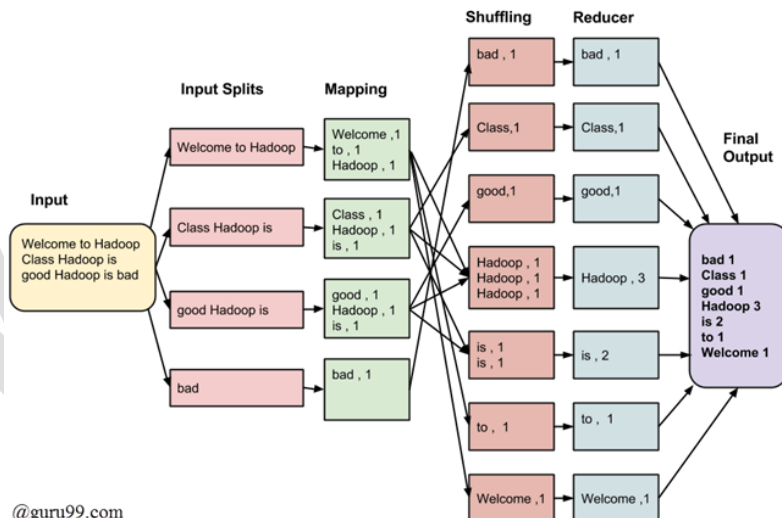
MapRed The job execution starts when the client program submit to the JobTracker a job configuration, which specifies the map, combine and reduce function, as well as the input and output path of data. The JobTracker will first determine the number of splits (each split is configurable, ~16-64MB) from the input path, and select some TaskTracker based on their network proximity to the data sources, then the JobTracker send the task requests to those selected TaskTrackers. Each TaskTracker will start the map phase processing by extracting the input data from the splits. For each record parsed by the “InputFormat”, it invoke the user provided “map” function, which emits a number of key/value pair in the memory buffer.

A periodic wakeup process will sort the memory buffer into different reducer node by invoke the “combine” function. The key/value pairs are sorted into one of the R local files (suppose there are R reducer nodes). When the map task completes (all splits are done), the

TaskTracker will notify the JobTracker. When all the TaskTrackers are done, the JobTracker will notify the selected TaskTrackers for the reduce phase. Each TaskTracker will read the region files remotely. It sorts the key/value pairs and for each key, it invoke the “reduce” function, which collects the key/aggregatedValue into the output file (one per reducer node).

Map/Reduce framework is resilient to crash of any components. The JobTracker keep tracks of the progress of each phases and periodically ping the TaskTracker for their health status. When any of the map phase TaskTracker crashes, the JobTracker will reassign the map task to a different TaskTracker node, which will rerun all the assigned splits. If the reduce phase TaskTracker crashes, the JobTracker will rerun the reduce at a different TaskTracker.

- Ex:
- Welcome to Hadoop Class
- Hadoop is good
- Hadoop is bad

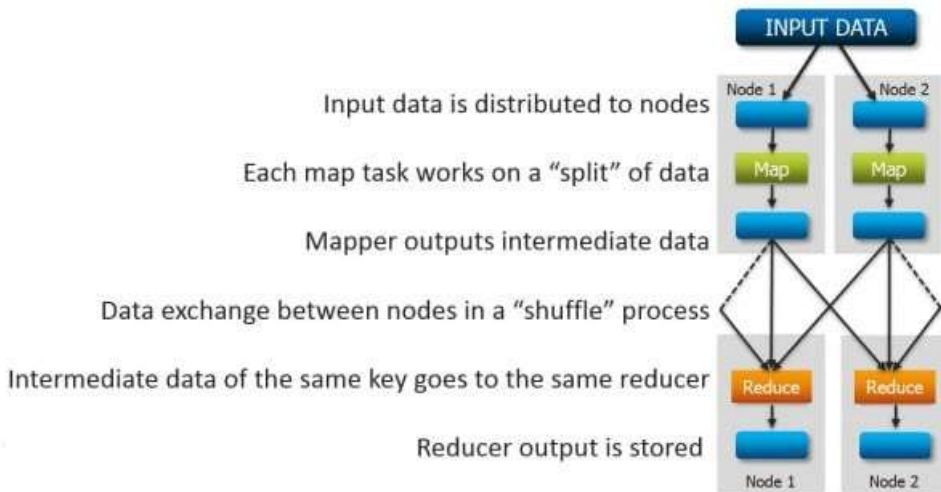


@guru99.com

- The data goes through following phases
- **Input Splits:**

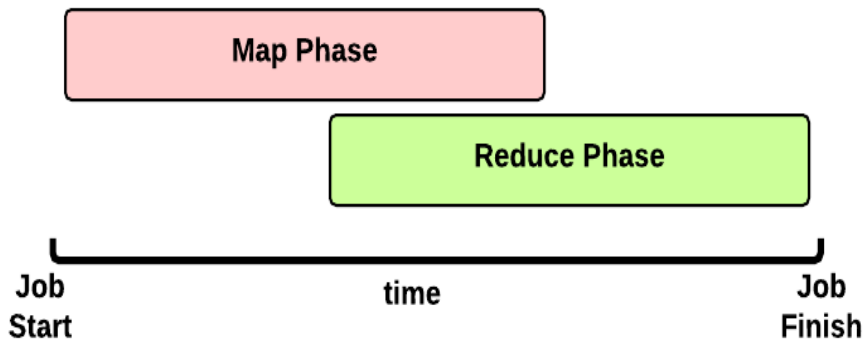
- Input to a MapReduce job is divided into fixed-size pieces called **input splits**. Input split is a chunk of the input that is consumed by a single map.
- **Mapping**
- This is the very first phase in the execution of map-reduce program. In this phase, data in each split is passed to a mapping function to produce output values. In our example, the job of the mapping phase is to count the number of occurrences of each word from input splits (more details about input-split are given below) and prepare a list in the form of <word, frequency>
- **Shuffling**
- This phase consumes the output of the Mapping phase. Its task is to consolidate the relevant records from the Mapping phase output. In our example, the same words are clubbed together along with their respective frequency.
- **Reducing**
- In this phase, output values from the Shuffling phase are aggregated. This phase combines values from the Shuffling phase and returns a single output value. In short, this phase summarizes the complete dataset.
bad 1
Class 1
good 1
Hadoop 3
is 2
to 1
Welcome 1

ANATOMY OF A MAPREDUCE JOB RUN



- Hadoop Framework comprises of two main components, namely, Hadoop Distributed File System (HDFS) for Data Storage and MapReduce for Data Processing.
- A typical Hadoop MapReduce job is divided into a set of Map and Reduce tasks that execute on a Hadoop cluster. The execution flow occurs as follows:
- Input data is split into small subsets of data.
- Map tasks work on these data splits.
- The intermediate input data from Map tasks is then submitted to Reduce task after an intermediate process called 'shuffle'.
- The Reduce task(s) works on this intermediate data to generate the result of a MapReduce Job.
- A job is divided into multiple tasks which are then run onto multiple data nodes in a cluster.
- It is the responsibility of jobtracker to coordinate the activity by **scheduling tasks** to run on different data nodes.
- Execution of individual task is then look after by tasktracker, which resides on every data node executing part of the job.

- Tasktracker's responsibility is to send the progress report to the jobtracker.
- In addition, tasktracker periodically sends '**heartbeat**' signal to the Jobtracker so as to notify him of current state of the system.
- Thus jobtracker keeps track of overall progress of each job. In the event of task failure, the jobtracker can reschedule it on a different tasktracker.



FAILURES

One of the major benefits of using Hadoop is its ability to handle such failures and allow your job to complete.

Task Failure

Consider first the case of the child task failing. The most common way that this happens is when user code in the map or reduce task throws a runtime exception. If this happens, the child JVM reports the error back to its parent tasktracker, before it exits. The error ultimately makes it into the user logs. The tasktracker marks the task attempt as failed, freeing up a slot to run another task.

For Streaming tasks, if the Streaming process exits with a nonzero exit code, it is marked as failed. This behavior is governed by the `stream.non.zero.exit.is.failure` property (the default is true).

Another failure mode is the sudden exit of the child JVM perhaps there is a JVM bug that causes the JVM to exit for a particular set of circumstances exposed by the Map- Reduce user code. In this case, the tasktracker notices that the process has exited and marks the attempt as failed.

Hanging tasks are dealt with differently. The tasktracker notices that it hasn't received a progress update for a while and proceeds to mark the task as failed. The child JVM process will be automatically killed after this period.* The timeout period after which tasks are considered failed is normally 10 minutes and can be configured on a per-job basis (or a cluster basis) by setting the `mapred.task.timeout` property to a value in milliseconds.

Setting the timeout to a value of zero disables the timeout, so long-running tasks are never marked as failed. In this case, a hanging task will never free up its slot, and over time there may be cluster slowdown as a result. This approach should therefore be avoided, and making sure that a task is reporting progress periodically will suffice (see “What Constitutes Progress in MapReduce?”).

When the jobtracker is notified of a task attempt that has failed (by the tasktracker's heartbeat call), it will reschedule execution of the task. The jobtracker will try to avoid rescheduling the task on a tasktracker where it has

previously failed. Furthermore, if a task fails four times (or more), it will not be retried further. This value is configurable: the maximum number of attempts to run a task is controlled by the `mapred.map.max.attempts` property for map tasks and `mapred.reduce.max.attempts` for reduce tasks. By default, if any task fails four times (or whatever the maximum number of attempts is configured to), the whole job fails.

For some applications, it is undesirable to abort the job if a few tasks fail, as it may be possible to use the results of the job despite some failures. In this case, the maximum percentage of tasks that are allowed to fail without triggering job failure can be set for the job. Map tasks and reduce tasks are controlled independently, using the `mapred.max.map.failures.percent` and `mapred.max.reduce.failures.percent` properties.

* If a Streaming process hangs, the tasktracker does not try to kill it (although the JVM that launched it will be killed), so you should take precautions to monitor for this scenario, and kill orphaned processes by some other means.

A task attempt may also be killed, which is different from it failing. A task attempt may be killed because it is a speculative duplicate (for more, see “Speculative Execution”), or because the tasktracker it was running on failed, and the jobtracker marked all the task attempts running on it as killed. Killed task attempts do not count against the number of attempts to run the task (as

set by `mapred.map.max.attempts` and `mapred.reduce.max.attempts`), since it wasn't the task's fault that an attempt was killed.

Users may also kill or fail task attempts using the web UI or the command line (type `hadoop job` to see the options). Jobs may also be killed by the same mechanisms. Tasktracker Failure

Failure of a tasktracker is another failure mode. If a tasktracker fails by crashing, or running very slowly, it will stop sending heartbeats to the jobtracker (or send them very infrequently). The jobtracker will notice a tasktracker that has stopped sending heartbeats (if it hasn't received one for 10 minutes, configured via the `mapred.task.tracker.expiry.interval` property, in milliseconds) and remove it from its pool of tasktrackers to schedule tasks on. The jobtracker arranges for map tasks that were run and completed successfully on that tasktracker to be rerun if they belong to incomplete jobs, since their intermediate output residing on the failed tasktracker's local filesystem may not be accessible to the reduce task. Any tasks in progress are also rescheduled.

A tasktracker can also be blacklisted by the jobtracker, even if the tasktracker has not failed. A tasktracker is blacklisted if the number of tasks that have failed on it is significantly higher than the average task failure rate on the cluster. Blacklisted tasktrackers can be restarted to remove them from the jobtracker's blacklist.

Jobtracker Failure

Failure of the jobtracker is the most serious failure mode. Currently, Hadoop has no mechanism for dealing with failure of the jobtracker it is a single point of failure so in this case the job fails. However, this failure mode has a low chance of occurring, since the chance of a particular machine failing is low. It is possible that a future release of Hadoop will remove this limitation by running multiple jobtrackers, only one of which is the primary jobtracker at any time

Shuffle and Sort

MapReduce makes the guarantee that the input to every reducer is sorted by key. The process by which the system performs the sort—and transfers the map outputs to the reducers as inputs—is known as the *shuffle*.⁴ In this section, we look at how the shuffle works, as a basic understanding will be helpful should you need to optimize a MapReduce program. The shuffle is an area of the codebase where refinements and improvements are continually being made, so the following description necessarily conceals many details. In many ways, the shuffle is the heart of MapReduce and is where the “magic” happens. The Map Side When the map function starts producing output, it is not simply written to disk. The process is more involved, and takes advantage of buffering writes in memory and doing some presorting for efficiency reasons.

Figure 7-4 shows what happens.

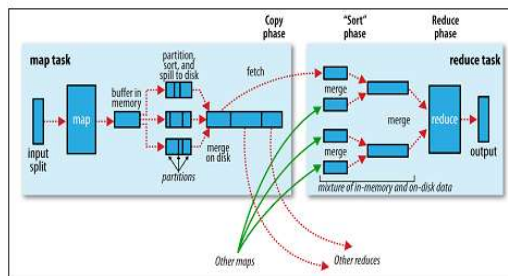


Figure 7-4. Shuffle and sort in MapReduce

Each map task has a circular memory buffer that it writes the output to. The buffer is 100 MB by default (the size can be tuned by changing the `mapreduce.task.io.sort.mb` property). When the contents of the buffer reach a certain threshold size (`mapreduce.map.sort.spill.percent`, which has the default value 0.80, or 80%), a background thread will start to *spill* the contents to disk. Map outputs will continue to be written to the buffer while the spill takes place, but if the buffer fills up during this time, the map will block until the spill is complete. Spills are written in round-robin fashion to the directories specified by the `mapreduce.cluster.local.dir` property, in a job-specific subdirectory. Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to.

Within each partition, the background thread performs an in-memory sort by key, and if there is a combiner function, it is run on the output of the sort. Running the combiner function makes for a more compact map output, so there is less data to write to local disk and to transfer to the reducer. Each time the memory buffer reaches the spill threshold, a new spill file is created, so after the map task has written its last output record, there could be several spill files. Before the task is finished, the spill files are merged into a single partitioned and sorted output file. The configuration property `mapreduce.task.io.sort.factor` controls the maximum number of streams to merge at once; the default is 10.

If there are at least three spill files (set by the `mapreduce.map.combine.minspills` property), the combiner is run again before the output file is written. Recall that combiners may be run repeatedly over the input without affecting the final result. If there are only one or two spills, the potential reduction in map output size is not worth the overhead in invoking the combiner, so it is not run again for this map output. It is often a good idea to compress the map output as it is written to disk, because doing so makes it faster to write to disk, saves disk space, and reduces the amount of data to transfer to the reducer. By default, the output is not compressed, but it is easy to enable this by setting `mapreduce.map.output.compress` to true.

The compression library to use is specified by `mapreduce.map.output.compress.codec`; for more on compression formats. The output file's partitions are made available to the reducers over HTTP.

The maximum number of worker threads used to serve the file partitions is controlled by the `mapreduce.shuffle.max.threads` property; this setting is per node manager, not per map task. The default of 0 sets the maximum number of threads to twice the number of processors on the machine.

The Reduce Side

Let's turn now to the reduce part of the process. The map output file is sitting on the local disk of the machine that ran the map task (note that although map outputs always get written to local disk, reduce outputs may not be), but now it is needed by the machine that is about to run the reduce task for the partition. Moreover, the reduce task needs the map output for its particular partition from several map tasks across the cluster. The map tasks may finish at different times, so the reduce task starts copying their outputs as soon as each completes. This is known as the *copy phase* of the reduce task. The reduce task has a small number of copier threads so that it can fetch map outputs in parallel. 198 | Chapter 7: How MapReduce Works The default is five threads, but this number can be changed by setting the `mapreduce.reduce.shuffle.parallelcopies` property. Map outputs are copied to the reduce task JVM's memory if they are small enough (the buffer's size is controlled by `mapreduce.reduce.shuffle.input.buffer.percent`, which specifies the proportion of the heap to use for this purpose); otherwise, they are copied to disk. When the in-memory buffer reaches a threshold size (controlled by `mapreduce.reduce.shuffle.merge.percent`) or reaches a threshold number of map outputs (`mapreduce.reduce.merge.inmem.threshold`), it is merged and spilled to disk. If a combiner is specified, it will be run during the merge to reduce the amount of data written to disk. As the copies accumulate on disk, a background thread merges them into larger, sorted files. This saves some time merging later on. Note that any map outputs that were compressed (by the map task) have to be decompressed in memory in order to perform a merge on them. When all the map outputs have been copied, the reduce task moves into the *sort phase* (which should properly be called the *merge phase*, as the sorting was carried out on the map side), which merges the map outputs, maintaining their sort ordering. This is done in rounds. For example, if there were 50 map outputs and the *merge factor* was 10 (the default, controlled by the

mapreduce.task.io.sort.factor property, just like in the map's merge), there would be five rounds.

Each round would merge 10 files into 1, so at the end there would be 5 intermediate files. Rather than have a final round that merges these five files into a single sorted file, the merge saves a trip to disk by directly feeding the reduce function in what is the last phase: the *reduce phase*. This final merge can come from a mixture of in-memory and on-disk segments.

TASK EXECUTION

We saw how the MapReduce system executes tasks in the context of the overall job at the beginning of this chapter, in “Anatomy of a MapReduce Job Run” on page 185. In this section, we’ll look at some more controls that MapReduce users have over task execution. TASK EXECUTION ENVIRONMENT Hadoop provides information to a map or reduce task about the environment in which it is running. For example, a map task can discover the name of the file it is processing (see “File information in the mapper” on page 227), and a map or reduce task can find out the attempt number of the task. The properties in Table 7-3 can be accessed from the job’s configuration, obtained in the old MapReduce API by providing an implementation of the configure() method for Mapper or Reducer, where the configuration is passed in as an argument. In the new API, these properties can be accessed from the context object passed to all methods of the Mapper or Reducer. Streaming environment variables Hadoop sets job configuration parameters as environment variables for Streaming programs. However, it replaces nonalphanumeric characters with underscores to make sure they are valid names. The following Python expression illustrates how you can retrieve the value of the mapreduce.job.id property from within a Python Streaming script: `os.environ["mapreduce_job_id"]`

Speculative Execution

The MapReduce model is to break jobs into tasks and run the tasks in parallel to make the overall job execution time smaller than it would be if the tasks ran sequentially. This makes the job execution time sensitive to slow-running tasks,

as it takes only one slow task to make the whole job take significantly longer than it would have done otherwise. When a job consists of hundreds or thousands of tasks, the possibility of a few straggling tasks is very real.

Tasks may be slow for various reasons, including hardware degradation or software misconfiguration, but the causes may be hard to detect because the tasks still complete successfully, albeit after a longer time than expected. Hadoop doesn't try to diagnose and fix slow-running tasks; instead, it tries to detect when a task is running slower than expected and launches another equivalent task as a backup. This is termed *speculative execution* of tasks. It's important to understand that speculative execution does not work by launching two duplicate tasks at about the same time so they can race each other. This would be wasteful of cluster resources. Rather, the scheduler tracks the progress of all tasks of the same type (map and reduce) in a job, and only launches speculative duplicates for the small proportion that are running significantly slower than the average. When a task completes successfully, any duplicate tasks that are running are killed since they are no longer needed. So, if the original task completes before the speculative task, the speculative task is killed; on the other hand, if the speculative task finishes first, the original is killed. Speculative execution is an optimization, and not a feature to make jobs run more reliably. If there are bugs that sometimes cause a task to hang or slow down, relying on speculative execution to avoid these problems is unwise and won't work reliably, since the same bugs are likely to affect the speculative task. You should fix the bug so that the task doesn't hang or slow down. Speculative execution is turned on by default. It can be enabled or disabled independently for map tasks and reduce tasks, on a cluster-wide basis, or on a per-job basis. The relevant properties are shown in [Table 7-4](#).

Property name	Type	Default value	Description
mapreduce.map.speculative	boolean	true	Whether extra instances of map tasks may be launched if a task is making slow progress
mapreduce.reduce.speculative	boolean	true	Whether extra instances of reduce tasks may be launched if a task is making slow progress
yarn.app.mapreduce.am.job.speculator.class	Class	org.apache.hadoop.mapreduce.v2.app.speculator.DefaultSpeculator	The Speculator class implementing the speculative execution policy (MapReduce 2 only)
yarn.app.mapreduce.am.job.task.estimator	Class	org.apache.hadoop.mapreduce.v2.app.spec	An implementation of TaskRuntimeEstimator used by Speculator

tor.class

late.LegacyTaskRunTimeEstimator

for instances that provides estimates
for task runtimes (MapReduce 2 only)

Why would you ever want to turn speculative execution off? The goal of speculative execution is to reduce job execution time, but this comes at the cost of cluster efficiency. On a busy cluster, speculative execution can reduce overall throughput, since redundant tasks are being executed in an attempt to bring down the execution time for a single job. For this reason, some cluster administrators prefer to turn it off on the cluster and have users explicitly turn it on for individual jobs.

MAP REDUCE TYPES AND FORMATS

MapReduce has a simple model of data processing: inputs and outputs for the map and reduce functions are key-value pairs. This chapter looks at the MapReduce model in detail, and in particular at how data in various formats, from simple text to structured binary objects, can be used with this model. MapReduce Types The map and reduce functions in Hadoop MapReduce have the following general form:

map:

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
    public class Context extends MapContext<KEYIN, VALUEIN,  
        KEYOUT, VALUEOUT> {  
        // ...  
    }  
    protected void map(KEYIN key, VALUEIN value,  
        Context context) throws IOException, InterruptedException {  
        // ...  
    }  
}  
  
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
    public class Context extends ReducerContext<KEYIN, VALUEIN,  
        KEYOUT, VALUEOUT> {  
        // ...  
    }  
    protected void reduce(KEYIN key, Iterable<VALUEIN> values,
```



```
Context context) throws IOException, InterruptedException {  
}  
}
```

The context objects are used for emitting key-value pairs, and they are parameterized

by the output types so that the signature of the write() method is:

```
public void write(KEYOUT key, VALUEOUT value)
```

throws IOException, InterruptedException

Since Mapper and Reducer are separate classes, the type parameters have different

scopes, and the actual type argument of KEYIN (say) in the Mapper may be different from

the type of the type parameter of the same name (KEYIN) in the Reducer.

For instance,

in the maximum temperature example from earlier chapters, KEYIN is replaced by Long

Writable for the Mapper and by Text for the Reducer.

Similarly, even though the map output types and the reduce input types must match,

this is not enforced by the Java compiler.

The type parameters are named differently from the abstract types

(KEYIN versus K1,

and so on), but the form is the same.

If a combiner function is used, then it has the same form as the reduce function (and is

an implementation of Reducer), except its output types are the intermediate key and

value types (K2 and V2), so they can feed the reduce function:

```
map: (K1, V1) → list(K2, V2)
```

```
combiner: (K2, list(V2)) → list(K2, V2)
```

```
reduce: (K2, list(V2)) → list(K3, V3)
```

Often the combiner and reduce functions are the same, in which case K3 is the same as

K2, and V3 is the same as V2.

The partition function operates on the intermediate key and value types (K2 and V2)

and returns the partition index. In practice, the partition is determined solely by the

key (the value is ignored):

partition: (K2, V2) → integer

Or in Java:

```
public abstract class Partitioner<KEY, VALUE> {  
    public abstract int getPartition(KEY key, VALUE value, int  
        numPartitions);}
```

The Default MapReduce Job

What happens when you run MapReduce without setting a mapper or a reducer? Let's

try it by running this minimal MapReduce program:

```
public class MinimalMapReduce extends Configured implements Tool {  
    @Override  
    public int run(String[] args) throws Exception {  
        if (args.length != 2) {  
            System.err.printf("Usage: %s [generic options] <input> <output>\n",  
                getClass().getSimpleName());  
            ToolRunner.printGenericCommandUsage(System.err);  
            return -1;  
        }  
        Job job = new Job(getConf());  
        job.setJarByClass(getClass());  
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  
        return job.waitForCompletion(true) ? 0 : 1;  
    }  
    public static void main(String[] args) throws Exception {  
        int exitCode = ToolRunner.run(new MinimalMapReduce(), args);  
        System.exit(exitCode);  
    }  
}
```

The only configuration that we set is an input path and an output path. We run it over

a subset of our weather data with the following:

% hadoop MinimalMapReduce "input/ncdc/all/190{1,2}.gz" output

We do get some output: one file named part-r-00000 in the output directory.

Here's what

the first few lines look like (truncated to fit the page):

0→0029029070999991901010106004+64333+023450FM-
12+000599999V0202701N01591...

0→0035029070999991902010106004+64333+023450FM-
12+000599999V0201401N01181...

135→0029029070999991901010113004+64333+023450FM-
12+000599999V0202901N00821...

141→0035029070999991902010113004+64333+023450FM-
12+000599999V0201401N01181...

270→0029029070999991901010120004+64333+023450FM-
12+000599999V0209991C00001...

282→0035029070999991902010120004+64333+023450FM-
12+000599999V0201401N01391...

Each line is an integer followed by a tab character, followed by the original weather data record. Admittedly, it's not a very useful program, but understanding how it produces its output does provide some insight into the defaults that Hadoop uses when running MapReduce jobs.

Example 8-1 shows a program that has exactly the same effect as MinimalMapReduce, but explicitly sets the job settings to their defaults.

The default Streaming job

In Streaming, the default job is similar, but not identical, to the Java equivalent. The basic form is:

```
% hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-  
streaming-*.jar \  
-input input/ncdc/sample.txt \  
-output output \  
-mapper /bin/cat
```

When we specify a non-Java mapper and the default text mode is in effect (-io text),

Streaming does something special. It doesn't pass the key to the mapper process; it just passes the value. (For other input formats, the same effect can be achieved by setting stream.map.input.ignoreKey to true.) This is

actually very useful because the key is just the line offset in the file and the value is the line, which is all most applications are interested in.

The overall effect of this job is to perform a sort of the input.

With more of the defaults spelled out, the command looks like this (notice that Streaming uses the old MapReduce API classes):

```
% hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar \  
-input input/ncdc/sample.txt \  
-output output \  
-inputformat org.apache.hadoop.mapred.TextInputFormat \  
-mapper /bin/cat \  
-partitioner org.apache.hadoop.mapred.lib.HashPartitioner \  
-numReduceTasks 1 \  
-reducer org.apache.hadoop.mapred.lib.IdentityReducer \  
-outputformat org.apache.hadoop.mapred.TextOutputFormat  
-io text
```

The `-mapper` and `-reducer` arguments take a command or a Java class. A combiner may optionally be specified using the `-combiner` argument. Keys and values in Streaming A Streaming application can control the separator that is used when a key-value pair is turned into a series of bytes and sent to the map or reduce process over standard input. The default is a tab character, but it is useful to be able to change it in the case that the keys or values themselves contain tab characters. Separators may be configured independently for maps and reduces. The properties are listed in Table 8-3 and shown in a diagram of the data flow path in Figure 8-1. These settings do not have any bearing on the input and output formats. For example, if `stream.reduce.output.field.separator` were set to be a colon, say, and the reduce stream process wrote the line `a:b` to standard out, the Streaming reducer would know to extract the key as `a` and the value as `b`. With the standard `TextOutputFormat`, this record would be written to the output file with a tab separating `a` and `b`. You can change the separator that `TextOutputFormat` uses by setting `mapreduce.output.textoutputformat.separator`.

MAP REDUCE FEATURES:

MapReduce is the framework that is used for processing large amounts of data on commodity hardware on a cluster ecosystem. The MapReduce is a powerful method of processing data when there are very huge amounts of node connected to the cluster. The two important tasks of the MapReduce algorithm are, as the name suggests – Map and Reduce.

The goal of the Map task is to take a large set of data and convert it into another set of data that is distinctly broken down into tuples or Key/Value pairs. Next the Reduce task takes the tuple which is the output of the Map task and makes the input for a reduction task. Here the data tuples are converted into a still smaller set of tuples. The Reduce task always follows the Map task.

The biggest strength of the MapReduce framework is scalability. Once a MapReduce program is written it can easily be extrapolated to work over a cluster which has hundreds or even thousands of nodes. In this framework, computation is sent to where the data resides.

The common terminology used in the MapReduce framework is as follows:

- **PayLoad:** both the Map and Reduce functions are implemented by the PayLoad applications which are the two most vital functions
- **Mapper:** the function of this application is to take the input/value pair and map it to a set of intermediate key/value pair
- **NameNode:** this is the node that is associated with HDFS
- **DataNode:** this is the node where the data is residing before the computation

- MasterNode: this is the node that takes job requests from the client and it is where the JobTracker runs
- SlaveNode: this is the node where both the Map and the Reduce tasks are run
- JobTracker: the jobs are scheduled here and the tracking of the jobs are reported here
- TaskTracker: it actually tracks the jobs and reports to the JobTracker with the status
- Task: it is the execution of the Mapper or the Reducer on a set of data

Counters

There are often things that you would like to know about the data you are analyzing but that are peripheral to the analysis you are performing. For example, if you were counting invalid records and discovered that the proportion of invalid records in the whole dataset was very high, you might be prompted to check why so many records were being marked as invalid—perhaps there is a bug in the part of the program that detects invalid records? Or if the data was of poor quality and genuinely did have very many invalid records, after discovering this, you might decide to increase the size of the dataset so that the number of good records was large enough for meaningful analysis.

Counters are a useful channel for gathering statistics about the job: for quality control or for application-level statistics. They are also useful for problem diagnosis. If you are tempted to put a log message into your map or reduce task, it is often better to see whether you can use a counter instead to record that a particular condition occurred. In addition to counter values being much easier to retrieve than log output for large distributed jobs, you get a record of the number of times that condition occurred, which is more work to obtain from a set of logfiles.

Built-in Counters

Hadoop maintains some built-in counters for every job, and these report various metrics. For example, there are counters for the number of bytes and records processed, which allow you to confirm that the expected amount of input was consumed and the expected amount of output was produced.

Counters are divided into groups, and there are several groups for the built-in counters, listed in [Table 9-1](#).

Table 9-1. Built-in counter groups

Group	Name/Enum	Reference
MapReduce task counters	<code>org.apache.hadoop.mapreduce.TaskCounter</code>	Table 9-2
Filesystem counters	<code>org.apache.hadoop.mapreduce.FileSystemCounter</code>	Table 9-3
FileInputFormat counters	<code>org.apache.hadoop.mapreduce.lib.input.FileInputFormatCounter</code>	Table 9-4
FileOutputFormat counters	<code>org.apache.hadoop.mapreduce.lib.output.FileOutputFormatCounter</code>	Table 9-5
Job counters	<code>org.apache.hadoop.mapreduce.JobCounter</code>	Table 9-6

Each group either contains *task counters* (which are updated as a task progresses) or *job counters* (which are updated as a job progresses). We look at both types in the following sections.

Task counters

Task counters gather information about tasks over the course of their execution, and the results are aggregated over all the tasks in a job. The `MAP_INPUT_RECORDS` counter, for example, counts the input records read by each map task and aggregates over all map tasks in a job, so that the final figure is the total number of input records for the whole job.

Task counters are maintained by each task attempt, and periodically sent to the application master so they can be globally aggregated. (This is described in [“Progress and Status Updates” on page 190](#).) Task counters are sent in full every time, rather than sending the counts since the last transmission, since this guards against errors due to lost messages. Furthermore, during a job run, counters may go down if a task fails.

Counter values are definitive only once a job has successfully completed. However, some counters provide useful diagnostic information as a task is progressing, and it can be useful to monitor them with the web UI. For

example, `PHYSICAL_MEMORY_BYTES`, `VIRTUAL_MEMORY_BYTES`, and `COMMITTED_HEAP_BYTES` provide an indication of how memory usage varies over the course of a particular task attempt.

The built-in task counters include those in the MapReduce task counters group (Table 9-2) and those in the file-related counters groups (Tables 9-3, 9-4, and 9-5).

Table 9-3. Built-in filesystem task counters

Counter	Description
<i>Filesystem</i> bytes read (<code>BYTES_READ</code>)	The number of bytes read by the filesystem by map and reduce tasks. There is a counter for each filesystem, and <i>Filesystem</i> may be Local, HDFS, S3, etc.
<i>Filesystem</i> bytes written (<code>BYTES_WRITTEN</code>)	The number of bytes written by the filesystem by map and reduce tasks.
<i>Filesystem</i> read ops (<code>READ_OPS</code>)	The number of read operations (e.g., open, file status) by the filesystem by map and reduce tasks.
<i>Filesystem</i> large read ops (<code>LARGE_READ_OPS</code>)	The number of large read operations (e.g., list directory for a large directory) by the filesystem by map and reduce tasks.
<i>Filesystem</i> write ops (<code>WRITE_OPS</code>)	The number of write operations (e.g., create, append) by the filesystem by map and reduce tasks.

Table 9-4. Built-in FileInputFormat task counters

Counter	Description
Bytes read (<code>BYTES_READ</code>)	The number of bytes read by map tasks via the <code>FileInputFormat</code> .

Table 9-5. Built-in FileOutputFormat task counters

Counter	Description
Bytes written (<code>BYTES_WRITTEN</code>)	The number of bytes written by map tasks (for map-only jobs) or reduce tasks via the <code>FileOutputFormat</code> .

Job counters

Job counters (Table 9-6) are maintained by the application master, so they don't need to be sent across the network, unlike all other counters, including user-defined ones. They measure job-level statistics, not values that change while a task is running. For

User-Defined Java Counters

MapReduce allows user code to define a set of counters, which are then incremented as desired in the mapper or reducer. Counters are defined by a Java enum, which serves to group related counters. A job may define an arbitrary number of enums, each with an arbitrary number of fields. The name of the enum is the group name, and the enum's fields are the counter names. Counters are global: the MapReduce framework aggregates them across all maps and reduces to produce a grand total at the end of the job.

We created some counters in **Chapter 6** for counting malformed records in the weather dataset. The program in **Example 9-1** extends that example to count the number of missing records and the distribution of temperature quality codes.

Dynamic counters

The code makes use of a dynamic counter—one that isn't defined by a Java enum. Because a Java enum's fields are defined at compile time, you can't create new counters on the fly using enums. Here we want to count the distribution of temperature quality codes, and though the format specification defines the values that the temperature quality code *can* take, it is more convenient to use a dynamic counter to emit the values that it *actually* takes. The method we use on the Context object takes a group and counter name using String names:

public Counter getCounter(String groupName, String counterName)

The two ways of creating and accessing counters—using enums and using strings are actually equivalent because Hadoop turns enums into strings to send counters over RPC. Enums are slightly easier to work with, provide type safety, and are suitable for most jobs. For the odd occasion when you need to create counters dynamically, you can use the String interface.

Retrieving counters

In addition to using the web UI and the command line (using `mapred job -counter`), you can retrieve counter values using the Java API. You can do

this while the job is running, although it is more usual to get counters at the end of a job run, when they are stable. **Example 9-2** shows a program that calculates the proportion of records that have missing temperature fields.

User-Defined Streaming Counters

A Streaming MapReduce program can increment counters by sending a specially formatted line to the standard error stream, which is co-opted as a control channel in this case. The line must have the following format:

```
reporter:counter:group,counter,amount
```

This snippet in Python shows how to increment the “Missing” counter in the “Temperature” group by 1:

```
sys.stderr.write("reporter:counter:Temperature,Missing,1\n")
```

In a similar way, a status message may be sent with a line formatted like this:

```
reporter:status:message
```

Sorting

The ability to sort data is at the heart of MapReduce. Even if your application isn’t concerned with sorting per se, it may be able to use the sorting stage that MapReduce provides to organize its data. In this section, we examine different ways of sorting datasets and how you can control the sort order in MapReduce.

Preparation

We are going to sort the weather dataset by temperature. Storing temperatures as Text objects doesn’t work for sorting purposes, because signed integers don’t sort lexicographically.¹ Instead, we are going to store the data using sequence files whose IntWritable keys represent the temperatures (and sort correctly) and whose Text values are the lines of data.

Partial Sort

In “**The Default MapReduce Job**” on page 214, we saw that, by default, MapReduce will sort input records by their keys. **Example 9-4** is a variation for sorting sequence files with IntWritable keys.

Total Sort

How can you produce a globally sorted file using Hadoop? The naive answer is to use a single partition.³ But this is incredibly inefficient for large files, because one machine has to process all of the output, so you are throwing away the benefits of the parallel architecture that MapReduce provides.

Instead, it is possible to produce a set of sorted files that, if concatenated, would form a globally sorted file. The secret to doing this is to use a partitioner that respects the total order of the output. For example, if we had four partitions, we could put keys for temperatures less than -10°C in the first partition, those between -10°C and 0°C in the second, those between 0°C and 10°C in the third, and those over 10°C in the fourth.

Secondary Sort

The MapReduce framework sorts the records by key before they reach the reducers. For any particular key, however, the values are *not* sorted. The order in which the values appear is not even stable from one run to the next, because they come from different map tasks, which may finish at different times from run to run. Generally speaking, most MapReduce programs are written so as not to depend on the order in which the values appear to the reduce function. However, it is possible to impose an order on the values by sorting and grouping the keys in a particular way.

To illustrate the idea, consider the MapReduce program for calculating the maximum temperature for each year. If we arranged for the values (temperatures) to be sorted in descending order, we wouldn’t have to iterate through them to find the maximum; instead, we could take the first for each year and ignore the rest. (This approach isn’t the most efficient way to solve this particular problem, but it illustrates how secondary sort works in general.)

Joins

MapReduce can perform joins between large datasets, but writing the code to do joins from scratch is fairly involved. Rather than writing MapReduce programs, you might consider using a higher-level framework such as Pig, Hive, Cascading, Cruc, or Spark, in which join operations are a core part of the implementation.

Let's briefly consider the problem we are trying to solve. We have two datasets—for example, the weather stations database and the weather records—and we want to reconcile the two. Let's say we want to see each station's history, with the station's metadata inlined in each output row. This is illustrated in **Figure 9-2**.

How we implement the join depends on how large the datasets are and how they are partitioned. If one dataset is large (the weather records) but the other one is small enough to be distributed to each node in the cluster (as the station metadata is), the join can be effected by a MapReduce job that brings the records for each station together (a partial sort on station ID, for example). The mapper or reducer uses the smaller dataset to look up the station metadata for a station ID, so it can be written out with each record. See **“Side Data Distribution” on page 273** for a discussion of this approach, where we focus on the mechanics of distributing the data to nodes in the cluster.

Map-Side Joins

A map-side join between large inputs works by performing the join before the data reaches the map function. For this to work, though, the inputs to each map must be partitioned and sorted in a particular way.

Reduce-Side Joins

A reduce-side join is more general than a map-side join, in that the input datasets don't have to be structured in any particular way, but it is less efficient because both datasets have to go through the MapReduce shuffle. The basic idea is that the mapper tags each record with its source and uses

the join key as the map output key, so that the records with the same key are brought together in the reducer.

POSSIBLE QUESTIONS

PART-B(5 x 6=30 Marks)

1. Explain about Scaling Out concept in hadoop
2. Explain about Task execution in detail
3. Describe Types of joins in detail
4. Explain about Sorting concept in detail
5. Explain about Map reduce types
6. Describe about Shuffle and Sort
7. Describe about Failures in hadoop

PART-C(1 x 10=10 Marks)

1. Explain about Anatomy of a Map Reduce Job run
2. Describe about Analyzing the Data with Hadoop
3. Explain about Hadoop Streaming in detail

WAVE



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS) (BATCH 2018-2020)

BIG DATA ANALYTICS

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

ONE MARK QUESTIONS

	UNIT-2	choice 1	choice 2	choice 3	choice 4			Answer
1	_____ license is Hadoop distributed under.	Apache License 2.0	Mozilla Public License	Shareware	Commercial			Apache License 2.0
2	Hadoop written in _____	Java (software platform)	Perl	Java (programming language)	Lua (programming language)			Java (programming language)
3	Hadoop run on _____	Bare metal	Debian	Cross-platform	Unix-like			Cross-platform
4	Hadoop achieves reliability by replicating the data across multiple hosts, and hence does not require _____ storage	RAID	Standard RAID levels	ZFS	Operating system			RAID
5	Above the file systems comes the _____ engine, which consists of one Job Tracker, to which client applications submit	MapReduce	Google	Functional programming	Facebook			MapReduce
6	HDFS supports the _____ command to fetch Delegation Token and store it in a file on the local system.	fetdt	fetchdt	fsk	rec			fetchdt
7	the NameNode will interactively prompt you at the command line about possible courses of action you can take to recover your data.	full	partial	recovery	commit			recovery
8	_____ command is used to copy file or directories recursively.	dtcp	distcp	dcp	distc			distcp

9	_____ mode is a Namenode state in which it does not accept changes to the name space.	Recover	Safe	Rollback	Partial			Rollback
10	used to interact and view Job Queue information in HDFS.	queue	priority	dist	Recover			queue
11	_____ is used for the MapReduce job Tracker node	mradmin	tasktrack er	jobtrack er	Admin			jobtrack er
12	master and there is only one NameNode per cluster.	Data Node	NameNo de	Data block	Replicati on			NameNo de
13	HDFS works in a _____ fashion.	master- worker	master- slave	worker/sl ave.	job tracker			master- worker
14	_____ NameNode is used when the Primary NameNode goes down.	Rack	Data	Secondar y	primary			Secondar y
15	slave/worker node and holds the user data in the form of Data Blocks.	DataNod e	NameNo de	Data block	Replicati on			DataNod e
16	HDFS provides a command line interface called _____ used to interact with HDFS.	“HDFS Shell”	“FS Shell”	“DFS Shell”	“ HS Shell”			“FS Shell”
17	_____ method clears all keys from the configuration.	clear	addReso urce	getClass	Add			clear
18	_____ adds a configuration resource	addReso urce	setDepre catedPro perties	addDefau ltResourc e	Resource			addReso urce
19	format is composed of approximately _____ blocks of compressed data.	128k	256k	24k	36k			256k
20	_____ is the default Partitioner for Mapreduce	MergePa rtitioner	HashedP artitioner	HashPart itioner	Hashing Partitione r			HashPart itioner

21	_____ partitions the key space	Partitioner	Comparator	Collector	full			Partitioner
22	_____ is a generalization of the facility provided by the MapReduce framework to collect data output by the Mapper or the Reducer	OutputCompressor	OutputCollector	InputCollector	Comparator			OutputCollector
23	_____ is the primary interface for a user to describe a MapReduce job to the Hadoop framework for execution.	JobConfig	JobConf	JobConfiguration	Configuration			JobConf
24	The _____ executes the Mapper/Reducer task as a child process in a separate jvm.	JobTracker	TaskTracker	TaskScheduler	Scheduler			JobTracker
25	Maximum virtual memory of the launched child-task is specified using :	mapv	mapred	mapvim	mapr			mapred
26	_____ is percentage of memory relative to the maximum heapsize in which map outputs may be retained during the reduce.	mapred.job.shuffle.merge.percent	mapred.job.reduce.input.buffer.percent	mapred.io.merge.threshold	io.sort.factor			mapred.job.reduce.input.buffer.percent
27	In order to read any file in HDFS, instance of _____ is required.	filesystem	datastream	outstream	inputstream			filesystem
28	_____ is used to read data from bytes buffers .	write()	read()	readwrite()	rewrite()			write()
29	reduces a set of intermediate values which share a key to a smaller set of values.	Mapper	Reducer	Writable	Readable			Reducer
30	_____ is input the grouped output of a	Mapper	Reducer	Writable	Readable			Mapper

31	Apache Hadoop's _____ provides a persistent data structure for binary key-value pairs.	GetFile	Sequence File	Putfile	copyfile			Sequence File
32	_____ formats of SequenceFile are present in Hadoop I/O	2	3	4	5			3
33	_____ format is more compression-aggressive	Partition Compressed	Record Compressed	Block-Compressed	Uncompressed			Block-Compressed
34	The _____ is a directory that contains two SequenceFile.	ReduceFile	MapperFile	MapFile	trackfile			MapFile
35	The _____ file is populated with the key and a LongWritable that contains the starting byte position of the record.	Array	Index	Immutable	mutable			Index
36	value field append(value) and the key is a LongWritable that contains the record number, count + 1.	SetFile	ArrayFile	BloomMapFile	unsetfile			ArrayFile
37	_____ data file takes is based on avro serializaton framework which was primarily created for hadoop.	Oozie	Avro	cTakes	Lucene			Avro
38	Avro schemas are defined with _____	JSON	XML	JAVA	HTML			JSON
39	_____ facilitates construction of generic data-processing systems and languages.	Untagged data	Dynamic typing	No manually-assigned field IDs	tagged data			Dynamic typing
40	With _____ we can store data and read it easily with various programming languages	Thrift	Protocol Buffers	Avro	Buffers			Avro
41	encoding structured data in an efficient yet extensible format.	Thrift	Protocol Buffers	Avro	Buffers			Protocol Buffers

42	Thrift resolves possible conflicts through _____ of the field.	Name	Static number	UID	dynamic number			Static number
43	future _____ layer of Hadoop.	RMC	RPC	RDC	RBC			RPC
44	We can declare the schema of our data either in a _____ file.	JSON	XML	SQL	R			SQL
45	Avro supports _____ kinds of complex types.	3	4	6	7			7
46	_____ are encoded as a series of blocks.	Arrays	Enum	Unions	Maps			Arrays
47	_____ encoded using the number of bytes declared in the schema.	Fixed	Enum	Unions	Maps			Fixed
48	_____ permits data written by one system to be efficiently sorted by another system.	Complex Data type	Order	Sort Order	Unsort Order			Sort Order
49	_____ between blocks to permit efficient splitting of files for MapReduce processing.	Codec	Data Marker	Synchroni zation markers	Unsyncro nization markers			Synchroni zation markers
50	The _____ codec uses Google's Snappy compression library.	null	snappy	deflate	delete			snappy
51	Avro messages are framed as a list of _____	buffers	frames	rows	columns			frames
52	_____ node is responsible for executing a Task assigned to it by the JobTracker	MapRed uce	Mapper	TaskTrac ker	JobTrack er			TaskTrac ker
53	_____ responsible for consolidating the results produced by each of the Map() functions/tasks.	Reduce	Map	Reducer	Unmap			Reduce
54	_____ which allows users to create and run jobs with any executable as the mapper and/or the reducer.	Hadoop Strdata	Hadoop Streamin g	Hadoop Stream	Hadoopd ata			Hadoop Streamin g

55	The number of maps is usually driven by the total size of _____	inputs	outputs	tasks	process			inputs
56	program involves running mapping tasks on many or all of the nodes in our cluster.	MapReduce	Map	Reducer	Reduce			MapReduce
57	data repository containing device information, images and other relevant information for all sorts of mobile devices.	DirectMemory	Directory	DeviceMap	Drill			DeviceMap
58	_____ is a secure and highly scalable microsharing and micromessaging platform.	ESME	Directory	Empire-db	Entity			ESME
59	used for building and consuming network services	ESME	Directory Map	Empire-db	Etch			Etch
60	open source system for expressive, declarative, fast, and efficient data analysis.	Flume	Flink	Flex	ESME			Flink

**UNIT III
SYLLABUS**

Hbase – data model and implementations – Hbase clients – Hbase examples – praxis. Cassandra – cassandra data model – cassandra examples – cassandra clients – Hadoop integration. Pig – Grunt – pig data model – Pig Latin – developing and testing Pig Latin scripts. Hive – data types and file formats – HiveQL data definition– HiveQL data manipulation

HBASE

HBase is a distributed column-oriented database built on top of HDFS. HBase is the Hadoop application to use when you require real-time read/write random access to very large datasets.

Although there are countless strategies and implementations for database storage and retrieval, most solutions—especially those of the relational variety—are not built with very large scale and distribution in mind. Many vendors offer replication and partitioning solutions to grow the database beyond the confines of a single node, but these add-ons are generally an afterthought and are complicated to install and maintain. They also severely compromise the RDBMS feature set. Joins, complex queries, triggers, views, and foreign-key constraints become prohibitively expensive to run on a scaled RDBMS, or do not work at all.

HBase approaches the scaling problem from the opposite direction. It is built from the ground up to scale linearly just by adding nodes. HBase is not relational and does not support SQL, but given the proper problem space, it is able to do what an RDBMS cannot: host very large, sparsely populated tables on clusters made from commodity hardware.

The canonical HBase use case is the *webtable*, a table of crawled web pages and their attributes (such as language and MIME type) keyed by the web page URL. The webtable is large, with row counts that run into the billions. Batch analytic and parsing.

BACKDROP

The HBase project was started toward the end of 2006 by Chad Walters and Jim Kellerman at Powerset. It was modeled after Google's Bigtable, which had just been published.² In February 2007, Mike Cafarella made a code drop of a mostly working system that Jim Kellerman then carried forward.

The first HBase release was bundled as part of Hadoop 0.15.0 in October 2007. In May 2010, HBase graduated from a Hadoop subproject to become an Apache Top Level Project. Today, HBase is a mature technology used in production across a wide range of industries.

DATA MODEL AND IMPLEMENTATIONS

Data Model

Applications store data in labeled tables. Tables are made of rows and columns. Table cells—the intersection of row and column coordinates—are versioned. By default, their version is a timestamp auto-assigned by HBase at the time of cell insertion. A cell's content is an uninterpreted array of bytes. An example HBase table for storing photos is shown in Figure 20-1.

Table row keys are also byte arrays, so theoretically anything can serve as a row key, from strings to binary representations of long or even serialized data structures. Table rows are sorted by row key, aka the table's primary key. The sort is byte-ordered. All table accesses are via the primary key.³

Row columns are grouped into *column families*. All column family members have a common prefix, so, for example, the columns `info:format` and `info:geo` are both members of the `info` column family, whereas `contents:image` belongs to the `contents` family. The column family prefix must be composed of *printable* characters. The qualifying tail, the column family *qualifier*, can be made of any arbitrary bytes. The column family and the qualifier are always separated by a colon character (:).

A table's column families must be specified up front as part of the table schema definition, but new column family members can be added on demand. For example, a new column `info:camera` can be offered by a client as part of an update, and its value persisted, as long as the column family `info` already exists on the table.

Physically, all column family members are stored together on the filesystem. So although earlier we described HBase as a column-oriented store, it would be more accurate if it were described as a column-*family*-oriented store. Because tuning and storage specifications are done at the column family level, it is advised that all column family members have the same general access pattern and size characteristics. For the photos table, the image data, which is large (megabytes), is stored in a separate column family from the metadata, which is much smaller in size (kilobytes).

In synopsis, HBase tables are like those in an RDBMS, only cells are versioned, rows are sorted, and columns can be added on the fly by the client as long as the column family they belong to preexists.

Regions

Tables are automatically partitioned horizontally by HBase into *regions*. Each region comprises a subset of a table's rows. A region is denoted by the table it belongs to, its first row (inclusive), and its last row (exclusive). Initially, a table comprises a single region, but as the region grows it eventually crosses a configurable size threshold, at which point it splits at a row boundary into two new regions of approximately equal size. Until this first split happens, all loading will be against the single server hosting the original region. As the table grows, the number of its regions grows. Regions are the units that get distributed over an HBase cluster. In this way, a table that is

too big for any one server can be carried by a cluster of servers, with each node hosting a subset of the table's total regions. This is also the means by which the loading on a table gets distributed. The online set of sorted regions comprises the table's total content.

Locking

Row updates are atomic, no matter how many row columns constitute the row-level transaction. This keeps the locking model simple.

Implementation

Just as HDFS and YARN are built of clients, workers, and a coordinating master—the *namenode* and *datanodes* in HDFS and *resource manager* and *node managers* in YARN—so is HBase made up of an HBase *master* node orchestrating a cluster of one or more *regionserver* workers (see Figure 20-2). The HBase master is responsible for bootstrapping a virgin install, for assigning regions to registered regionservers, and for recovering regionserver failures. The master node is lightly loaded. The regionservers carry zero or more regions and field client read/write requests. They also manage region splits, informing the HBase master about the new daughter regions so it can manage the offlining of parent regions and assignment of the replacement daughters.

HBase depends on ZooKeeper (Chapter 21), and by default it manages a ZooKeeper instance as the authority on cluster state, although it can be configured to use an existing ZooKeeper cluster instead. The ZooKeeper ensemble hosts vitals such as the location of the `hbase:meta` catalog table and the address of the current cluster master. Assignment of regions is mediated via ZooKeeper in case participating servers crash mid-assignment. Hosting the assignment transaction state in ZooKeeper makes it so recovery can pick up on the assignment where the crashed server left off. At a minimum, when bootstrapping a client connection to an HBase cluster, the client must be passed the location of the ZooKeeper ensemble. Thereafter, the client navigates the ZooKeeper hierarchy to learn cluster attributes such as server locations.

Regionserver worker nodes are listed in the HBase `conf/regionservers` file, as you would list `datanodes` and `node managers` in the Hadoop `etc/hadoop/slaves` file. Start and stop scripts are like those in Hadoop and use the same SSH-based mechanism for running remote commands. A cluster's site-specific configuration is done in the HBase `conf/hbase-site.xml` and `conf/hbase-env.sh` files, which have the same format as their equivalents in the Hadoop parent project. HBase persists data via the Hadoop filesystem API. Most people using HBase run it on HDFS for storage, though by default, and unless told otherwise, HBase writes to the local filesystem. The local filesystem is fine for experimenting with your initial HBase install, but thereafter, the first configuration made in an HBase cluster usually involves pointing HBase at the HDFS cluster it should use.

HBase in operation

Internally, HBase keeps a special catalog table named `hbase:meta`, within which it maintains the current list, state, and locations of all user-space regions afloat on the cluster. Entries in `hbase:meta` are keyed by region name, where a region name is made up of the name of the table the region belongs to, the region's start row, its time of creation, and finally, an MD5 hash of all of these (i.e., a hash of table name, start row, and creation timestamp). Here is an example region name for a region in the table `TestTable` whose start row is `xyz`:

TestTable,xyz,1279729913622.1b6e176fb8d8aa88fd4ab6bc80247ece.

Commas delimit the table name, start row, and timestamp. The MD5 hash is surrounded by a leading and trailing period.

As noted previously, row keys are sorted, so finding the region that hosts a particular row is a matter of a lookup to find the largest entry whose key is less than or equal to that of the requested row key. As regions transition—are split, disabled, enabled, deleted, or redeployed by the region load balancer, or redeployed due to a regionserver crash—the catalog table is updated so the state of all regions on the cluster is kept current.

Fresh clients connect to the ZooKeeper cluster first to learn the location of hbase:meta. The client then does a lookup against the appropriate hbase:meta region to figure out the hosting user-space region and its location. Thereafter, the client interacts directly with the hosting regionserver.

To save on having to make three round-trips per row operation, clients cache all they learn while doing lookups for hbase:meta. They cache locations as well as user-space region start and stop rows, so they can figure out hosting regions themselves without having to go back to the hbase:meta table. Clients continue to use the cached entries as they work, until there is a fault. When this happens—i.e., when the region has moved the client consults the hbase:meta table again to learn the new location. If the consulted hbase:meta region has moved, then ZooKeeper is reconsulted.

Writes arriving at a regionserver are first appended to a commit log and then added to an in-memory *memstore*. When a memstore fills, its content is flushed to the filesystem.

The commit log is hosted on HDFS, so it remains available through a regionserver crash. When the master notices that a regionserver is no longer reachable, usually because the server's znode has expired in ZooKeeper, it splits the dead regionserver's commit log by region. On reassignment and before they reopen for business, regions that were on the dead regionserver will pick up their just-split files of not-yet-persisted edits and replay them to bring themselves up to date with the state they had just before the failure.

When reading, the region's memstore is consulted first. If sufficient versions are found reading memstore alone, the query completes there. Otherwise, flush files are consulted in order, from newest to oldest, either until versions sufficient to satisfy the query are found or until we run out of flush files.

A background process compacts flush files once their number has exceeded a threshold, rewriting many files as one, because the fewer files a read consults, the more performant it will be. On compaction, the process cleans out versions beyond the schema-configured maximum and removes deleted and expired cells. A separate process running in the regionserver monitors flush file sizes, splitting the region when they grow in excess of the configured maximum.

HBASE CLIENTS

There are a number of client options for interacting with an HBase cluster.

Java

HBase, like Hadoop, is written in Java. Example 20-1 shows the Java version of how you would do the shell operations listed in the previous section.

Example 20-1. Basic table administration and access

```
public class ExampleClient {

    public static void main(String[] args) throws IOException { Configuration
        config = HBaseConfiguration.create();
        // Create table
        HBaseAdmin admin = new HBaseAdmin(config);
        try {
            TableName tableName = TableName.valueOf("test");
            HTableDescriptor htd = new HTableDescriptor(tableName);
            HColumnDescriptor hcd = new HColumnDescriptor("data");
            htd.addFamily(hcd);
            admin.createTable(htd);
            HTableDescriptor[] tables = admin.listTables(); if
            (tables.length != 1 &&
                Bytes.equals(tableName.getName(), tables[0].getTableName().getName())) { throw
                new IOException("Failed create of table");
            }
            // Run some operations -- three puts, a get, and a scan -- against the table.
            HTable table = new HTable(config, tableName); try {

            for (int i = 1; i <= 3; i++) {

            byte[] row = Bytes.toBytes("row" + i);

            Put put = new Put(row);

            byte[] columnFamily = Bytes.toBytes("data");

            byte[] qualifier = Bytes.toBytes(String.valueOf(i)); byte[] value = Bytes.toBytes("value" + i);
            put.add(columnFamily, qualifier, value); table.put(put);
            }

            Get get = new Get(Bytes.toBytes("row1"));

            Result result = table.get(get);

            System.out.println("Get: " + result);

            Scan scan = new Scan();

            ResultScanner scanner = table.getScanner(scan); try {
```



```

for (Result scannerResult : scanner) {

    System.out.println("Scan: " + scannerResult);

}

} finally {

    scanner.close();

}

// Disable then drop the table admin.disableTable(tableName); admin.deleteTable(tableName);

} finally { table.close();
}

} finally { admin.close();
}

}

}

```

This class has a main() method only. For the sake of brevity, we do not include the package name, nor imports. Most of the HBase classes are found in the org.apache.hadoop.hbase and org.apache.hadoop.hbase.client packages.

In this class, we first ask the HBaseConfiguration class to create a Configuration object. It will return a Configuration that has read the HBase configuration from the hbase-site.xml and hbase-default.xml files found on the program's classpath. This Configuration is subsequently used to create instances of HBaseAdmin and HTable. HBaseAdmin is used for administering your HBase cluster, specifically for adding and dropping tables. HTable is used to access a specific table. The Configuration instance points these classes at the cluster the code is to work against.

To create a table, we need to create an instance of HBaseAdmin and then ask it to create the table named test with a single column family named data. In our example, our table schema is the default. We could use methods on HTableDescriptor and HColumn Descriptor to change the table schema. Next, the code asserts the table was actually created, and throws an exception if it wasn't.

To operate on a table, we will need an instance of HTable, which we construct by passing it our Configuration instance and the name of the table. We then create Put objects in a loop to insert data into the table. Each Put puts a single cell value of value_{*n*} into a row named row_{*n*} on the column named data:_{*n*}, where *n* is from 1 to 3. The column name is specified in two parts: the column family name, and the column family qualifier. The code makes liberal use of HBase's Bytes utility class (found in the org.apache.hadoop.hbase.util package) to convert identifiers and values to the byte arrays that HBase requires.

Next, we create a Get object to retrieve and print the first row that we added. Then we use a Scan object to scan over the table, printing out what we find.

At the end of the program, we clean up by first disabling the table and then deleting it (recall that a table must be disabled before it can be dropped).

The simplest way to compile the program is to use the Maven POM that comes with the book's example code. Then we can use the hbase command followed by the classname to run the program. Here's a sample run:

```
1.mvn package
2.export HBASE_CLASSPATH=hbase-examples.jar
3.hbase ExampleClient
Get: keyvalues={row1/data:1/1414932826551/Put/vlen=6/mvcc=0}
Scan: keyvalues={row1/data:1/1414932826551/Put/vlen=6/mvcc=0}
Scan: keyvalues={row2/data:2/1414932826564/Put/vlen=6/mvcc=0}
Scan: keyvalues={row3/data:3/1414932826566/Put/vlen=6/mvcc=0}
```

Each line of output shows an HBase row, rendered using the toString() method from Result. The fields are separated by a slash character, and are as follows: the row name, the column name, the cell timestamp, the cell type, the length of the value's byte array (vlen), and an internal HBase field (mvcc). We'll see later how to get the value from a Result object using its getValue() method.

MapReduce

HBase classes and utilities in the org.apache.hadoop.hbase.mapreduce package facilitate using HBase as a source and/or sink in MapReduce jobs. The TableInputFormat class makes splits on region boundaries so maps are handed a single region to work on. The TableOutputFormat will write the result of the reduce into HBase.

The SimpleRowCounter class in Example 20-2 (which is a simplified version of Row Counter in the HBase mapreduce package) runs a map task to count rows using TableInputFormat.

The RowCounterMapper nested class is a subclass of the HBase TableMapper abstract class, a specialization of org.apache.hadoop.mapreduce.Mapper that sets the map inputs and types passed by TableInputFormat. Input keys are ImmutableBytesWritable objects (row keys), and values are Result objects (row results from a scan). Since this job counts rows and does not emit any output from the map, we just increment Counters.ROWS by 1 for every row we see.

In the run() method, we create a scan object that is used to configure the job by invoking the TableMapReduceUtil.initTableMapJob() utility method, which, among other things (such as setting the map class to use), sets the input format to TableInputFormat.

Notice how we set a filter, an instance of FirstKeyOnlyFilter, on the scan. This filter instructs the server to short-circuit when running server-side, populating the Result object in the mapper with only the first cell in each row. Since the mapper ignores the cell values, this is a useful optimization.

REST and Thrift

HBase ships with REST and Thrift interfaces. These are useful when the interacting application is written in a language other than Java. In both cases, a Java server hosts an instance of the

HBase client brokering REST and Thrift application requests into and out of the HBase cluster. Consult the Reference Guide for information on running the services, and the client interfaces.

PRAXIS

HDFS

HBase's use of HDFS is very different from how it's used by MapReduce. In MapReduce, generally, HDFS files are opened with their content streamed through a map task and then closed. In HBase, datafiles are opened on cluster startup and kept open so that we avoid paying the costs associated with opening files on each access. Because of this, HBase tends to see issues not normally encountered by MapReduce clients:

Running out of file descriptors

Because we keep files open, on a loaded cluster it doesn't take long before we run into system- and Hadoop-imposed limits. For instance, say we have a cluster that has three nodes, each running an instance of a datanode and a regionserver, and we're running an upload into a table that is currently at 100 regions and 10 column families. Allow that each column family has on average two flush files. Doing the math, we can have $100 \times 10 \times 2$, or 2,000, files open at any one time. Add to this total other miscellaneous descriptors consumed by outstanding scanners and Java libraries. Each open file consumes at least one descriptor over on the remote data node.

The default limit on the number of file descriptors per process is 1,024. When we exceed the filesystem *ulimit*, we'll see the complaint about "Too many open files" in logs, but often we'll first see indeterminate behavior in HBase. The fix requires increasing the file descriptor *ulimit* count; 10,240 is a common setting. Consult the HBase Reference Guide for how to increase the *ulimit* on your cluster.

Running out of datanode threads

Similarly, the Hadoop datanode has an upper bound on the number of threads it can run at any one time. Hadoop 1 had a low default of 256 for this setting (*dfs.datanode.max.xcievers*), which would cause HBase to behave erratically. Hadoop 2 increased the default to 4,096, so you are much less likely to see a problem for recent versions of HBase (which only run on Hadoop 2 and later). You can change the setting by configuring *dfs.datanode.max.transfer.threads* (the new name for this property) in *hdfs-site.xml*.

UI

HBase runs a web server on the master to present a view on the state of your running cluster. By default, it listens on port 60010. The master UI displays a list of basic attributes such as software versions, cluster load, request rates, lists of cluster tables, and participating regionserver. Click on a regionserver in the master UI, and you are taken to the web server running on the individual regionserver. It lists the regions this server is carrying and basic metrics such as resources consumed and request rates.

Metrics

Hadoop has a metrics system that can be used to emit vitals over a period to a *context* (this is covered in "Metrics and JMX" on page 331). Enabling Hadoop metrics, and in particular tying them to Ganglia or emitting them via JMX, will give you views on what is happening on your cluster, both currently and in the recent past. HBase also adds metrics of its own—request rates,

counts of vitals, resources used. See the file *hadoop-metrics2-hbase.properties* under the HBase *conf* directory.

Counters

At StumbleUpon, the first production feature deployed on HBase was keeping counters for the *stumbleupon.com* frontend. Counters were previously kept in MySQL, but the rate of change was such that drops were frequent, and the load imposed by the counter writes was such that web designers self imposed limits on what was counted. Using the `incrementColumnValue()` method on `HTable`, counters can be incremented many thousands of times a second.

CASSANDRA

Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. It is a type of NoSQL database. Let us first understand what a NoSQL database does.

NoSQLDatabase

A NoSQL database (sometimes called as Not Only SQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, eventually consistent, and can handle huge amounts of data.

The primary objective of a NoSQL database is to have

- simplicity of design,
- horizontal scaling, and
- finer control over availability.

NoSql databases use different data structures compared to relational databases. It makes some operations faster in NoSQL. The suitability of a given NoSQL database depends on the problem it must solve.

NoSQL vs. Relational Database

The following table lists the points that differentiate a relational database from a NoSQL database.

Relational Database	NoSql Database
Supports powerful query language.	Supports very simple query language.

It has a fixed schema.	No fixed schema.
Follows ACID (Atomicity, Consistency, Isolation, and Durability).	It is only “eventually consistent”.
Supports transactions.	Does not support transactions.

Besides Cassandra, we have the following NoSQL databases that are quite popular:

- Apache HBase - HBase is an open source, non-relational, distributed database modeled after Google’s BigTable and is written in Java. It is developed as a part of Apache Hadoop project and runs on top of HDFS, providing BigTable-like capabilities for Hadoop.
- MongoDB - MongoDB is a cross-platform document-oriented database system that avoids using the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas making the integration of data in certain types of applications easier and faster.

What is Apache Cassandra?

Apache Cassandra is an open source, distributed and decentralized/distributed storage system (database), for managing very large amounts of structured data spread out across the world. It provides highly available service with no single point of failure.

Listed below are some of the notable points of Apache Cassandra:

- It is scalable, fault-tolerant, and consistent.
- It is a column-oriented database.
- Its distribution design is based on Amazon’s Dynamo and its data model on Google’s Bigtable.
- Created at Facebook, it differs sharply from relational database management systems.
- Cassandra implements a Dynamo-style replication model with no single point of failure, but adds a more powerful “column family” data model.
- Cassandra is being used by some of the biggest companies such as Facebook, Twitter, Cisco, Rackspace, ebay, Twitter, Netflix, and more.

Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- Elastic scalability - Cassandra is highly scalable; it allows to add more hardware to accommodate more customers and more data as per requirement.
- Always on architecture - Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- Fast linear-scale performance - Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- Flexible data storage - Cassandra accommodates all possible data formats including: structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- Easy data distribution - Cassandra provides the flexibility to distribute data where you need by replicating data across multiple data centers.
- Transaction support - Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- Fast writes - Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

History of Cassandra

- Cassandra was developed at Facebook for inbox search.
- It was open-sourced by Facebook in July 2008.
- Cassandra was accepted into Apache Incubator in March 2009.
- It was made an Apache top-level project since February 2010.

CASSANDRA DATAMODEL

The data model of Cassandra is significantly different from what we normally see in an RDBMS. This chapter provides an overview of how Cassandra stores its data.

Cluster

Cassandra database is distributed over several machines that operate together. The outermost container is known as the Cluster. For failure handling, every node contains a replica, and in

case of a failure, the replica takes charge. Cassandra arranges the nodes in a cluster, in a ring format, and assigns data to them.

Keyspace

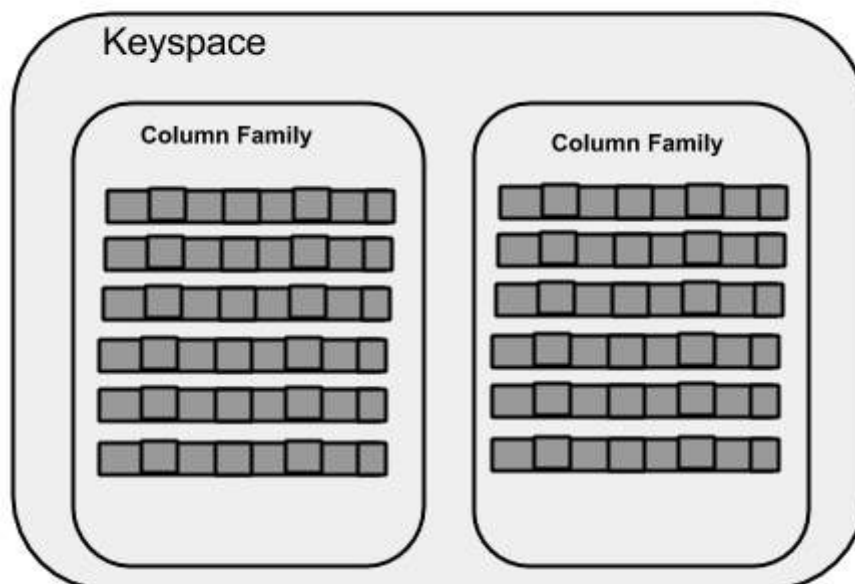
Keyspace is the outermost container for data in Cassandra. The basic attributes of a Keyspace in Cassandra are –

- Replication factor – It is the number of machines in the cluster that will receive copies of the same data.
- Replica placement strategy – It is nothing but the strategy to place replicas in the ring. We have strategies such as simple strategy (rack-aware strategy), old network topology strategy (rack-aware strategy), and network topology strategy (datacenter-shared strategy).
- Column families – Keyspace is a container for a list of one or more column families. A column family, in turn, is a container of a collection of rows. Each row contains ordered columns. Column families represent the structure of your data. Each keyspace has at least one and often many column families.

The syntax of creating a Keyspace is as follows –

```
CREATE KEYSPACE Keyspace name  
WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

The following illustration shows a schematic view of a Keyspace.



Column Family

A column family is a container for an ordered collection of rows. Each row, in turn, is an ordered collection of columns. The following table lists the points that differentiate a column family from a table of relational databases.

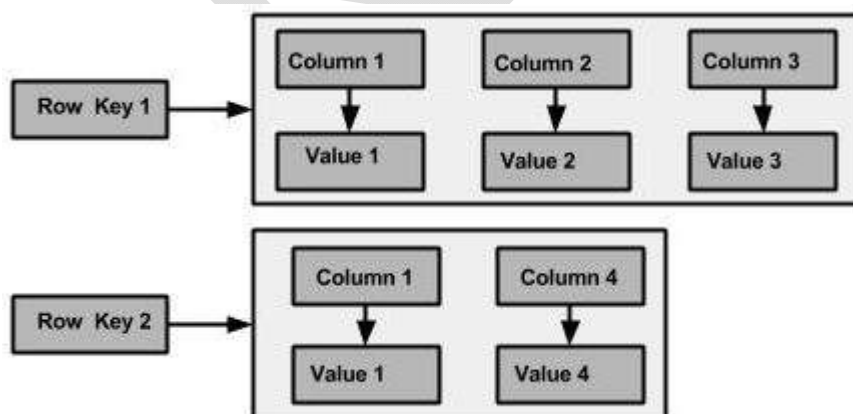
Relational Table	Cassandra column Family
A schema in a relational model is fixed. Once we define certain columns for a table, while inserting data, in every row all the columns must be filled at least with a null value.	In Cassandra, although the column families are defined, the columns are not. You can freely add any column to any column family at any time.
Relational tables define only columns and the user fills in the table with values.	In Cassandra, a table contains columns, or can be defined as a super column family.

A Cassandra column family has the following attributes –

- `keys_cached` – It represents the number of locations to keep cached per SSTable.
- `rows_cached` – It represents the number of rows whose entire contents will be cached in memory.
- `preload_row_cache` – It specifies whether you want to pre-populate the row cache.

Note – Unlike relational tables where a column family's schema is not fixed, Cassandra does not force individual rows to have all the columns.

The following figure shows an example of a Cassandra column family.



Column

A column is the basic data structure of Cassandra with three values, namely key or column name, value, and a time stamp. Given below is the structure of a column.

Column		
name : byte[]	value : byte[]	clock : clock[]

SuperColumn

A super column is a special column, therefore, it is also a key-value pair. But a super column stores a map of sub-columns.

Generally column families are stored on disk in individual files. Therefore, to optimize performance, it is important to keep columns that you are likely to query together in the same column family, and a super column can be helpful here. Given below is the structure of a super column.

Super Column	
name : byte[]	cols : map<byte[], column>

DATA MODELS OF CASSANDRA AND RDBMS

The following table lists down the points that differentiate the data model of Cassandra from that of an RDBMS.

RDBMS	Cassandra
RDBMS deals with structured data.	Cassandra deals with unstructured data.
It has a fixed schema.	Cassandra has a flexible schema.
In RDBMS, a table is an array of arrays. (ROW x COLUMN)	In Cassandra, a table is a list of “nested key-value pairs”. (ROW x COLUMN key x COLUMN value)
Database is the outermost container that contains data corresponding to an	Keyspace is the outermost container that

application.	contains data corresponding to an application.
Tables are the entities of a database.	Tables or column families are the entity of a keyspace.
Row is an individual record in RDBMS.	Row is a unit of replication in Cassandra.
Column represents the attributes of a relation.	Column is a unit of storage in Cassandra.
RDBMS supports the concepts of foreign keys, joins.	Relationships are represented using collections.

CASSANDRA CLIENT

Cassandra support for integrating Hadoop with Cassandra includes:

- MapReduce
- Apache Pig

You can use Cassandra 2.1 with Hadoop 2.x or 1.x with some restrictions.

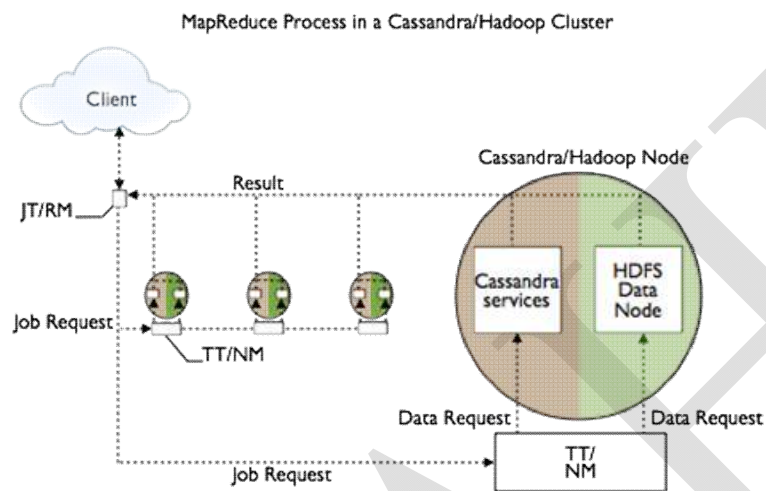
- Isolate Cassandra and Hadoop nodes in separate datacenters.
- Before starting the datacenters of Cassandra/Hadoop nodes, disable virtual nodes (vnodes).

To disable virtual nodes:

1. In the *cassandra.yaml* file, set num_tokens to 1.
2. Uncomment the initial_token property and set it to 1 or to the value of a generated token for a multi-node cluster.
3. Start the cluster for the first time.

Do not disable or enable vnodes on an existing cluster.

Setup and configuration, involves overlaying a Hadoop cluster on Cassandra nodes, configuring a separate server for the Hadoop NameNode/JobTracker, and installing a Hadoop TaskTracker and Data Node on each Cassandra node. The nodes in the Cassandra datacenter can draw from data in the HDFS Data Node as well as from Cassandra. The Job Tracker/Resource Manager (JT/RM) receives MapReduce input from the client application. The JT/RM sends a MapReduce job request to the Task Trackers/Node Managers (TT/NM) and optional clients, MapReduce and Pig. The data is written to Cassandra and results sent back to the client.



The Apache docs also cover how to get configuration and integration support.

Input and Output Formats

Hadoop jobs can receive data from CQL tables and indexes and you can load data into Cassandra from a Hadoop job. Cassandra 2.1 supports the following formats for these tasks:

- CQL partition input format: ColumnFamilyInputFormat class.
- BulkOutputFormat class introduced in Cassandra 1.1

Cassandra 2.1.1 and later supports the CqlOutputFormat, which is the CQL-compatible version of the BulkOutputFormat class. The CqlOutputFormat acts as a Hadoop-specific OutputFormat. Reduce tasks can store keys (and corresponding bound variable values) as CQL rows (and respective columns) in a given CQL table.

Cassandra 2.1.1 supports using the CqlOutputFormat with Apache Pig.

Running the wordcount example

Wordcount example JARs are located in the examples directory of the Cassandra source code installation. There are CQL and legacy examples in the `hadoop_cql3_word_count` and `hadoop_word_count` subdirectories, respectively. Follow instructions in the readme files.

Isolating Hadoop and Cassandra workloads

When you create a keyspace using CQL, Cassandra creates a virtual datacenter for a cluster, even a one-node cluster, automatically. You assign nodes that run the same type of workload to the same datacenter. The separate, virtual datacenters for different types of nodes segregate workloads running Hadoop from those running Cassandra. Segregating workloads ensures that only one type of workload is active per datacenter. Separating nodes running a sequential data load, from nodes running any other type of workload, such as Cassandra real-time OLTP queries is a best practice.

PIG :

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Pig.

The language used to analyze data in Hadoop using Pig is known as Pig Latin. It is a highlevel data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output.

Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.

Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

GRUNT

Grunt is a JavaScript Task Runner which can be used as a command line tool for JavaScript objects. It is a task manager written on top of NodeJS.

Why Use Grunt?

- Grunt can perform repetitive tasks very easily, such as compilation, unit testing, minifying files, running tests, etc.
- Grunt includes built-in tasks that extend the functionality of your plugins and scripts.
- The ecosystem of Grunt is huge; you can automate anything with very less effort.

History

The first lines of source code were added to GruntJS in 2011. The Grunt v0.4 was released on February 18, 2013. The Grunt v0.4.5 was released on May 12, 2014. The stable version of Grunt is 1.0.0 rc1 which was released on February 11, 2016.

Advantages

- Using Grunt, you can perform minification, compilation, and testing of files easily.
- Grunt unifies the workflows of web developers.
- You can easily work with a new codebase using Grunt because it contains less infrastructure.
- It speeds up the development workflow and enhances the performance of projects.

Disadvantages

- Whenever npm packages are updated, you need to wait until the author of the Grunt updates it.
- Every task is designed to do a specified work. If you want to extend a specified task, then you need to use some tricks to get the work done.
- Grunt includes a large number of configuration parameters for individual plugins. Usually, Grunt configuration files are longer in length.

PIG LATIN DATA MODEL

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as map and tuple. Given below is the diagrammatical representation of Pig Latin's data model.

Atom

Any single value in Pig Latin, irrespective of their data, type is known as an Atom. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a field.

Example – 'raja' or '30'

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Example – (Raja, 30)

Bag

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by '{}'. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example – {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as inner bag.

Example – {Raja, 30, {9848022338, raja@gmail.com,}}

Map

A map (or data map) is a set of key-value pairs. The key needs to be of type chararray and should be unique. The value might be of any type. It is represented by '['']'

Example – [name#Raja, age#30]

Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

PIG LATIN

To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.

Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.

Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.

Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

DEVELOPING AND TESTING PIG LATIN SCRIPTS

Comments in Pig Script

While writing a script in a file, we can include comments in it as shown below.

Multi-line comments

We will begin the multi-line comments with '/*', end them with '*'.

```
/* These are the multi-line comments
```

```
  In the pig script */
```

Single –line comments

We will begin the single-line comments with '--'.

```
--we can write single line comments like this.
```

Executing Pig Script in Batch mode

While executing Apache Pig statements in batch mode, follow the steps given below.

Step 1

Write all the required Pig Latin statements in a single file. We can write all the Pig Latin statements and commands in a single file and save it as .pig file.

Step 2

Execute the Apache Pig script. You can execute the Pig script from the shell (Linux) as shown below.

Executing a Pig Script from HDFS

We can also execute a Pig script that resides in the HDFS. Suppose there is a Pig script with the name Sample_script.pig in the HDFS directory named /pig_data/. We can execute it as shown below.

```
$ pig -x mapreduce hdfs://localhost:9000/pig_data/Sample_script.pig
```

Example

Assume we have a file student_details.txt in HDFS with the following content.

HIVE

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Hive is not

A relational database

A design for OnLine Transaction Processing (OLTP)

A language for real-time queries and row-level updates

Features of Hive

It stores schema in a database and processed data into HDFS.

It is designed for OLAP.

It provides SQL type language for querying called HiveQL or HQL.

It is familiar, fast, scalable, and extensible.

Architecture of Hive

The following component diagram depicts the architecture of Hive:

Hive Architecture

This component diagram contains different units. The following table describes each unit:

Unit Name	Operation
-----------	-----------

User Interface Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).

Meta Store Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.

HiveQL Process Engine HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.

Execution Engine The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.

HDFS or HBASE Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

DATATYPES AND FILE FORMATS:

All the data types in Hive are classified into four types, given as follows:

Column Types

Literals

Null Values

Complex Types

Column Types

Column type are used as column data types of Hive. They are as follows:

Integral Types

Integer type data can be specified using integral data types, INT. When the data range exceeds the range of INT, you need to use BIGINT and if the data range is smaller than the INT, you use SMALLINT. TINYINT is smaller than SMALLINT.

The following table depicts various INT data types:

Type	Postfix	Example
TINYINT	Y	10Y
SMALLINT	S	10S
INT	-	10
BIGINT	L	10L

String Types

String type data types can be specified using single quotes (' ') or double quotes (" "). It contains two data types: VARCHAR and CHAR. Hive follows C-types escape characters.

The following table depicts various CHAR data types:

Data Type Length
VARCHAR 1 to 65535
CHAR 255

Timestamp

It supports traditional UNIX timestamp with optional nanosecond precision. It supports java.sql.Timestamp format “YYYY-MM-DD HH:MM:SS.ffffffff” and format “yyyy-mm-dd hh:mm:ss.ffffffff”.

Dates

DATE values are described in year/month/day format in the form {{YYYY-MM-DD}}.

Decimals

The DECIMAL type in Hive is as same as Big Decimal format of Java. It is used for representing immutable arbitrary precision. The syntax and example is as follows:

DECIMAL(precision, scale)

decimal(10,0)

Union Types

Union is a collection of heterogeneous data types. You can create an instance using create union. The syntax and example is as follows:

UNIONTYPE<int, double, array<string>, struct<a:int,b:string>>

{0:1}

{1:2.0}

{2:["three","four"]}

{3:{ "a":5,"b":"five" }}

{2:["six","seven"]}

{3:{ "a":8,"b":"eight" }}

{0:9}

{1:10.0}

Literals

The following literals are used in Hive:

Floating Point Types

Floating point types are nothing but numbers with decimal points. Generally, this type of data is composed of DOUBLE data type.

Decimal Type

Decimal type data is nothing but floating point value with higher range than DOUBLE data type. The range of decimal type is approximately -10³⁰⁸ to 10³⁰⁸.

Null Value

Missing values are represented by the special value NULL.

Complex Types

The Hive complex data types are as follows:

Arrays

Arrays in Hive are used the same way they are used in Java.

Syntax: ARRAY<data_type>

Maps

Maps in Hive are similar to Java Maps.

Syntax: MAP<primitive_type, data_type>

Structs

Structs in Hive is similar to using complex data with comment.

Syntax: STRUCT<col_name : data_type [COMMENT col_comment], ...>

The conventions of creating a table in HIVE is quite similar to creating a table using SQL.

HIVEQL DATA DEFINITION

Create Table Statement

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.] table_name

[(col_name data_type [COMMENT col_comment], ...)]

[COMMENT table_comment]

[ROW FORMAT row_format]

[STORED AS file_format]

Example

Let us assume you need to create a table named employee using CREATE TABLE statement.

The following table lists the fields and their data types in employee table:

Sr.No	Field Name	Data Type
1	Eid	int
2	Name	String

3	Salary	Float
4	Designation	string

The following data is a Comment, Row formatted fields such as Field terminator, Lines terminator, and Stored File type.

```
COMMENT 'Employee details'
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED IN TEXT FILE
```

The following query creates a table named employee using the above data.

```
hive> CREATE TABLE IF NOT EXISTS employee ( eid int, name String,
salary String, destination String)
COMMENT 'Employee details'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;
```

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

On successful creation of table, you get to see the following response:

```
OK
Time taken: 5.905 seconds
hive>
```

JDBC Program

The JDBC program to create a table is given example.

```
import java.sql.SQLException;

import java.sql.Connection;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.DriverManager;
```

```
public class HiveCreateTable {  
  
    private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";  
  
    public static void main(String[] args) throws SQLException {  
  
        // Register driver and create driver instance  
        Class.forName(driverName);  
  
        // get connection  
        Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "",  
        "");  
  
        // create statement  
        Statement stmt = con.createStatement();  
  
        // execute statement  
        stmt.executeQuery("CREATE TABLE IF NOT EXISTS "  
            +" employee ( eid int, name String, "  
            +" salary String, destignation String)"  
            +" COMMENT 'Employee details'"  
            +" ROW FORMAT DELIMITED"  
            +" FIELDS TERMINATED BY '\t'"  
            +" LINES TERMINATED BY '\n'"  
            +" STORED AS TEXTFILE;");  
    }  
}
```

```
System.out.println(" Table employee created.");  
  
con.close();  
  
}  
  
}
```

Save the program in a file named HiveCreateDb.java. The following commands are used to compile and execute this program.

```
$ javac HiveCreateDb.java  
$ java HiveCreateDb
```

Output

Table employee created.

Load Data Statement

Generally, after creating a table in SQL, we can insert data using the Insert statement. But in Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

Syntax

The syntax for load data is as follows:

```
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename  
[PARTITION (partcol1=val1, partcol2=val2 ...)]
```

- LOCAL is identifier to specify the local path. It is optional.
- OVERWRITE is optional to overwrite the data in the table.
- PARTITION is optional.

Example

We will insert the following data into the table. It is a text file named sample.txt in /home/user directory.

```
1201 Gopal    45000  Technical manager  
1202 Manisha  45000  Proof reader  
1203 Masthanvali 40000  Technical writer  
1204 Kiran    40000  Hr Admin
```

1205 Kranthi 30000 Op Admin

The following query loads the given text into the table.

```
hive> LOAD DATA LOCAL INPATH '/home/user/sample.txt'  
OVERWRITE INTO TABLE employee;
```

On successful download, you get to see the following response:

```
OK  
Time taken: 15.905 seconds  
hive>
```

JDBC Program

Given below is the JDBC program to load given data into the table.

```
import java.sql.SQLException;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.Statement;  
import java.sql.DriverManager;  
  
public class HiveLoadData {  
  
    private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";  
  
    public static void main(String[] args) throws SQLException {  
  
        // Register driver and create driver instance  
        Class.forName(driverName);  
  
        // get connection
```

```
Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "",
");

// create statement

Statement stmt = con.createStatement();

// execute statement

stmt.executeQuery("LOAD DATA LOCAL INPATH '/home/user/sample.txt' +
"OVERWRITE INTO TABLE employee;");

System.out.println("Load Data into employee successful");

con.close();
}
}
```

Save the program in a file named HiveLoadData.java. Use the following commands to compile and execute this program.

```
$ javac HiveLoadData.java
$ java HiveLoadData
```

Output:

```
Load Data into employee successful
```

Alter Table Statement

It is used to alter a table in Hive.

Syntax

The statement takes any of the following syntaxes based on what attributes we wish to modify in a table.

ALTER TABLE name RENAME TO new_name

ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])

ALTER TABLE name DROP [COLUMN] column_name

ALTER TABLE name CHANGE column_name new_name new_type

ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])

Rename To... Statement

The following query renames the table from employee to emp.

```
hive> ALTER TABLE employee RENAME TO emp;
```

Change Statement

The following table contains the fields of employee table and it shows the fields to be changed (in bold).

Field Name	Convert from Data Type	Change Field Name	Convert to Data Type
eid	int	eid	int
name	String	ename	String
salary	Float	salary	Double
designation	String	designation	String

The following queries rename the column name and column data type using the above data:

```
hive> ALTER TABLE employee CHANGE name ename String;  
hive> ALTER TABLE employee CHANGE salary salary Double;
```

Drop Table Statement

The syntax is as follows:

```
DROP TABLE [IF EXISTS] table_name;
```

The following query drops a table named employee:

```
hive> DROP TABLE IF EXISTS employee;
```

On successful execution of the query, you get to see the following response:

```
OK
```

```
Time taken: 5.3 seconds
```

hive>

JDBC Program

The following JDBC program drops the employee table.

```
import java.sql.SQLException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.DriverManager;

public class HiveDropTable {

    private static String driverName = "org.apache.hadoop.hive.jdbc.HiveDriver";

    public static void main(String[] args) throws SQLException {

        // Register driver and create driver instance
        Class.forName(driverName);

        // get connection
        Connection con = DriverManager.getConnection("jdbc:hive://localhost:10000/userdb", "",
        "");

        // create statement
        Statement stmt = con.createStatement();

        // execute statement
```

```
stmt.executeQuery("DROP TABLE IF EXISTS employee;");

System.out.println("Drop table successful.");

con.close();

}

}
```

Save the program in a file named HiveDropTable.java. Use the following commands to compile and execute this program.

```
$ javac HiveDropTable.java
$ java HiveDropTable
```

Output:

```
Drop table successful
```

The following query is used to verify the list of tables:

```
hive> SHOW TABLES;

emp

ok

Time taken: 2.1 seconds

hive>
```

HIVEQL QUERIES

Hive Query language (HiveQL) provides SQL type environment in Hive to work with tables, databases, queries.

- Order by query
- Group by query
- Sort by
- Cluster By
- Distribute By

ORDER BY

- The ORDER BY syntax in HiveQL is similar to the syntax of ORDER BY in SQL language.

- Order by is the clause we use with "SELECT" statement in Hive queries, which helps sort data. Order by clause use columns on Hive tables for sorting particular column values mentioned with Order by. For whatever the column name we are defining the order by clause the query will select and display results by ascending or descending order the particular column values.
- If the mentioned order by field is a string, then it will display the result in lexicographical order. At the back end, it has to be passed on to a single reducer.

GROUP BY

Group by clause use columns on Hive tables for grouping particular column values mentioned with the group by. For whatever the column name we are defining a "groupby" clause the query will select and display results by grouping the particular column values.

Sort by:

Sort by clause performs on column names of Hive tables to sort the output. We can mention DESC for sorting the order in descending order and mention ASC for Ascending order of the sort.

In this sort by it will sort the rows before feeding to the reducer. Always sort by depends on column types.

For instance, if column types are numeric it will sort in numeric order if the columns types are string it will sort in lexicographical order.

Cluster By:

Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL.

Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by to distribute the rows among reducers. Cluster BY columns will go to the multiple reducers.

- It ensures sorting orders of values present in multiple reducers

For example, Cluster By clause mentioned on the Id column name of the table employees_guru table. The output when executing this query will give results to multiple reducers at the back end. But as front end it is an alternative clause for both Sort By and Distribute By.

This is actually back end process when we perform a query with sort by, group by, and cluster by in terms of Map reduce framework. So if we want to store results into multiple reducers, we go with Cluster By.

Distribute By:

Distribute BY clause used on tables present in Hive. Hive uses the columns in Distribute by to distribute the rows among reducers. All Distribute BY columns will go to the same reducer.

- It ensures each of N reducers gets non-overlapping ranges of column
- It doesn't sort the output of each reducer

POSSIBLE QUESTIONS

(5 x 6 = 30 MARKS)

1. Explain about Hbase clients
2. Describe about data types in hive in detail
3. Explain about Map reduce types
4. Explain about Praxis

(1 x 10 =10 MARKS)

1. Describe about data model and implementations in hbase
2. Explain Cassandra data model in detail



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS) (BATCH 2018-2020)

BIG DATA ANALYTICS

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

ONE MARK QUESTIONS

	UNIT-3	choice 1	choice 2	choice 3	choice 4			Answer
1	The framework groups Reducer inputs by key in _____ stage.	sort	shuffle	reduce	unsort			sort
2	The output of the reduce task is typically written to the FileSystem via _____ .	OutputCollector.collect	OutputCollector.get	OutputCollector.receive	OutputCollector.put			OutputCollector.collect
3	_____ can change the maximum number of cells of a column family.	set	reset	alter	select			alter
4	_____ is not a table scope operator	MEMSTORE_FLUSH	MEMSTORE_FLUSHSIZE	MAX_FLUSHSIZE	MEMSTORE_FLUSHSIZE			MEMSTORE_FLUSH
5	You can delete a column family from a table using the method _____ of HBaseAdmin class.	deleteColumn()	removeColumn()	deleteColumn()	delete			deleteColumn()
6	_____ class adds HBase configuration files to its object.	Configuration	Collector	Component	Compiler			Configuration
7	The _____ class provides the getValue() method to read the values from its instance.	Get	Result	Put	Value			Result
8	_____ communicate with the client and handle data-related operations.	Master Server	Region Server	Htable	Rtable			Region Server
9	_____ is the main configuration file of HBase.	hbase.xml	hbase-site.xml	hbase-site-conf.xml	hbase-site.config			hbase-site.xml
10	HBase uses the _____ File System to store its data.	Hive	Imphala	Hadoop	Scala			Hadoop

11	_____ is project for Infrastructure Engineers and Data Scientists	Impala	BigTop	Oozie	Flume			BigTop
12	_____ operating system is not supported by BigTop	Fedora	Solaris	Ubuntu	SUSE			Solaris
13	Apache Bigtop uses _____ for continuous integration testing.	Jenkinstop	Jerry	Jenkins	James			Jenkins
14	The Bigtop Jenkins server runs daily jobs for the _____ and trunk branches.	0.1	0.2	0.3	0.4			0.3
15	_____ builds an APT or YUM package repository	Bigtop-trunk-package-test	Bigtop-trunk-repository	Bigtop-VM-matrix	Bigtop-trunk			Bigtop-trunk-repository
16	Ambari leverages _____ for metrics collection.	Nagios	Nagaond	Ganglia	Nahoida			Ganglia
17	Ambari leverages _____ for system alerting and will send emails when your attention is needed	Nagios	Nagaond	Ganglia	Nahoida			Nagios
18	A _____ is a way of extending Ambari that allows 3rd parties to plug in new resource types along with the APIs	trigger	view	schema	semantic s			view
19	Ambari _____ deliver a template approach to cluster deployment.	View	Stack Advisor	Blueprint s	schema			Blueprint s
20	_____ facilitates installation of Hadoop across any number of hosts.	API-driven installations	Wizard-driven interface	Extensible framework	Nahoida			Wizard-driven interface

21	Ambari provides a _____ API that enables integration with existing tools, such as Microsoft System Center	RestLess	Web Service	RESTful	Web Application			RESTful
22	If Ambari Agent has any output in /var/log/ambari-agent/ambari-agent.out, it is indicative of a _____ problem.	Less Severe	Significant	Extremely Severe	mitigate			Significant
23	Avro schemas describe the format of the message and are defined using :	JSON	XML	JS	BSON			JSON
24	The _____ is an iterator which reads through the file and returns objects using the next() method.	DatReader	DatumReader	DatumReader	Data			DatumReader
25	The _____ class extends and implements several Hadoop-supplied interfaces.	AvroReducer		AvroMapper	Avro			AvroMapper
26	The _____ method in the ModelCountReducer class “reduces” the values the mapper collects into a derived value	count	add	reduce	alter			reduce
27	_____ works well with Avro	Lucene	kafka	MapReduce	DesignPattern			MapReduce
28	_____ tools is used to generate proxy objects in Java to easily work with the objects.	Lucene	kafka	MapReduce	Avro			Avro
29	The minimum number of row versions to keep is configured per column family via :	HBaseDescriptor	HTableDescriptor.	HColumnDescriptor	Hbase			HColumnDescriptor
30	HBase supports a _____ interface via Put and Result	“bytes-in/bytes-out”	“bytes-in”	“bytes-out”	"bytes"			“bytes-in/bytes-out”

31	One supported datatype that deserves special mention are :	money	counters	smallint	tinyint			counters
32	The _____ suffers from the monotonically increasing rowkey problem	rowkey	columnkey	counterkey	shardkey			rowkey
33	_____ does re-write data and pack rows into columns for certain time-periods.	OpenTS	OpenTSDB	OpenTSD	OpenDB			OpenTSDB
34	_____ command is used to disable all the tables matching the given regex	remove all	drop all	disable_all	enable_all			disable_all
35	_____ command disables drops and recreates a table.	drop	truncate	delete	start			truncate
36	_____ interface is implemented by Sqoop for recording	SqoopWrite	SqoopRecord	SqoopRead	Sqoop			SqoopRecord
37	Sqoop is an open source tool written at _____	Cloudera	IBM	Microsoft	Polaris			Microsoft
38	Sqoop uses _____ to fetch data from RDBMS and stores that on HDFS.	Hive	Map reduce	Imphala	BigTOP			Map reduce
39	_____ allows users to specify the target location inside of Hadoop.	Imphala	Oozie	Sqoop	Hive			Sqoop
40	Microsoft uses a Sqoop-based connector to help transfer data from _____ databases to Hadoop	PostgreSQL	SQL Server	Oracle	MySQL			SQL Server
41	_____ provides a Couchbase Server-Hadoop connector by means of Sqoop.	MemCache	Couchbase	Hbase	Hive			MemCache
42	Sqoop's direct mode does not support imports of _____ columns	BLOB	LONGVARBINARY	CLOB	SLOB			CLOB

43	Sqoop has been tested with Oracle _____ Express Edition.	11.2.0	10.2.0	12.2.0	10.3.0			10.2.0
44	_____ tool can list all the available database schemas.	sqoop-list-tables	sqoop-list-databases	sqoop-list-schema	sqoop-list-columns			sqoop-list-databases
45	Data can be imported in maximum _____ file formats.	1	2	3	5			2
46	_____ text is appropriate for most non-binary data types.	Character	Binary	Delimited	Decimal			Delimited
47	If you set the inline LOB limit to ____, all large objects will be placed in external storage.	0	1	2	3			0
48	_____ does not support the notion of enclosing characters that may include field delimiters in the enclosed string.	Imphala	Oozie	Sqoop	Hive			Hive
49	Sqoop can also import the data into Hive by generating and executing a _____ statement to define the data's layout in Hive.	SET TABLE	CREATE TABLE	INSERT TABLE	UNSET TABLE			CREATE TABLE
50	The _____ tool imports a set of tables from an RDBMS to HDFS.	export-all-tables	import-all-tables	import-tables	import			import-tables
51	_____ argument is not supported by import-all-tables tool	-class-name	-package-name	-database-name	-table-name			-class-name
52	The type of _____ strategy Cassandra performs on your data is configurable and can significantly affect read performance.	compression	collection	compaction	decompression			compaction

53	There are _____ types of read requests that a coordinator can send to a replica	two	three	four	five			three
54	_____ can be configured per table for non-QUORUM consistency levels	Read repair	Read damage	Write repair	Write damage			Read repair
55	If the table has been configured with the _____ property, the coordinator node for the read request will retry the request with another replica node .	rapid_retry	speculative_retry	speculative_rapid	None of the mentioned			speculative_retry
56	_____ operating system is not supported by BigTop	Fedora	Solaris	Ubuntu	SUSE			Solaris
57	Apache Bigtop uses _____ for continuous integration testing.	Jenkins	Jerry	Jenkins	None of the mentioned			Jenkins
58	The Apache Jenkins server runs the _____ job whenever code is committed to the trunk branch.	"Bigtop-trunk"	"Bigtop"	"Big-trunk"	"Big"			"Bigtop-trunk"
59	The Bigtop Jenkins server runs daily jobs for the _____ and trunk branches.	0.1	0.2	0.3	0.4			0.3
60	_____ builds an APT or YUM package repository	Bigtop-trunk-package-test	Bigtop-trunk-repository	Bigtop-VM-matrix	Bigtop-trunk			Bigtop-trunk-repository

UNIT IV
SYLLABUS

Introduction to NoSQL – aggregate data models – aggregates – key-value and document data models – relationships– schemaless databases – materialized views – distribution models -peer-peer replication –consistency – relaxing consistency – version stamps – partitioning and combining – composing map-reduce calculations -Document based Database – MongoDB- Introduction- Data Model- Working with data- Replication & Sharding- Development

INTRODUCTION TO NOSQL:

The term NoSQL was coined by Carlo Strozzi in the year 1998. He used this term to name his Open Source, Light Weight, DataBase which did not have an SQL interface.

In the early 2009, when last.fm wanted to organize an event on open-source distributed databases, Eric Evans, a Rackspace employee, reused the term to refer databases which are non-relational, distributed, and does not conform to atomicity, consistency, isolation, durability - four obvious features of traditional relational database systems.

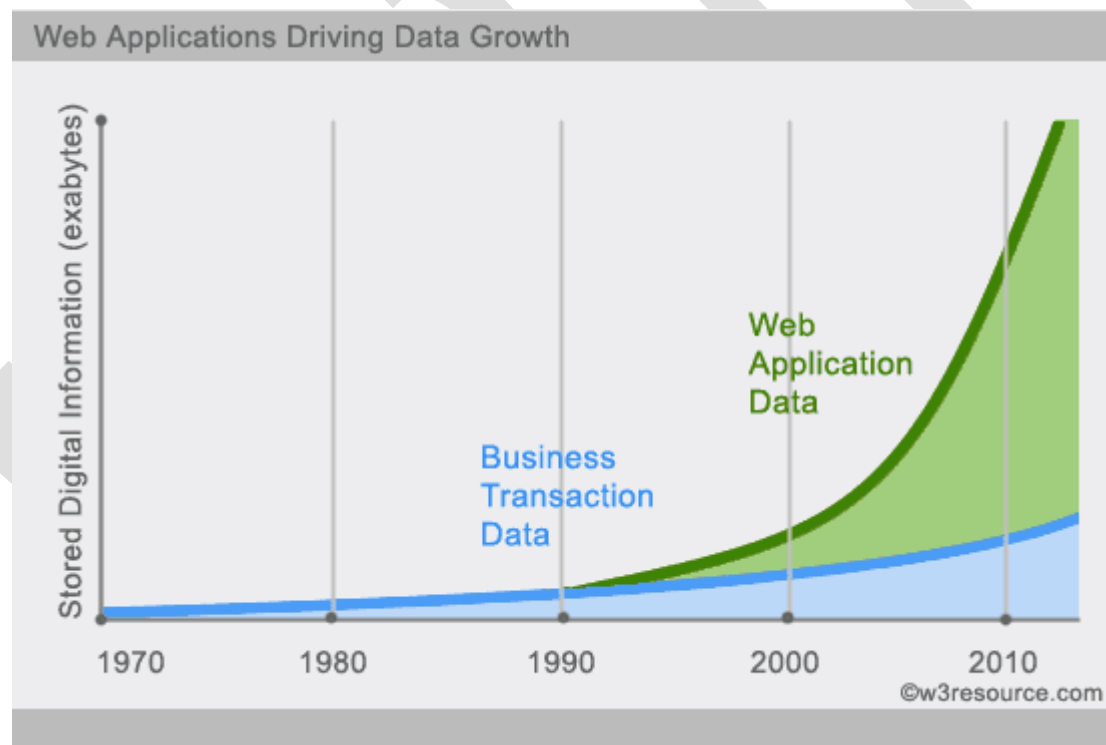
In the same year, the "no:sql(east)" conference held in Atlanta, USA, NoSQL was discussed and debated a lot.

And then, discussion and practice of NoSQL got a momentum, and NoSQL saw an unprecedented growth.

NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways. It is designed for distributed data stores where very large scale of data storing needs (for example Google or Facebook which collects terabits of data every day for their users). These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

Why NoSQL?

In today's time data is becoming easier to access and capture through third parties such as Facebook, Google+ and others. Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially. To avail the above service properly, it is required to process huge amount of data. Which SQL databases were never designed. The evolution of NoSql databases is to handle these huge data properly.



Example :

Social-network graph:

Each record: UserID1, UserID2

Separate records: UserID, first_name,last_name, age, gender,...

Task: Find all friends of friends of friends of ... friends of a given user.

Wikipedia pages :

Large collection of documents

Combination of structured and unstructured data

Task: Retrieve all pages regarding athletics of Summer Olympic before 1950.

AGGREGATE DATA MODELS:

Relational database modelling is vastly different than the types of data structures that application developers use. Using the data structures as modelled by the developers to solve different problem domains has given rise to movement away from relational modelling and towards aggregate models, most of this is driven by *Domain Driven Design*, a book by Eric Evans. An aggregate is a collection of data that we interact with as a unit. These units of data or aggregates form the boundaries for ACID operations with the database, Key-value, Document, and Column-family databases can all be seen as forms of aggregate-oriented database.

Aggregates make it easier for the database to manage data storage over clusters, since the unit of data now could reside on any machine and when retrieved from the database gets all the related data along with it. Aggregate-oriented databases work best when most data interaction is done with the same aggregate, for example when there is need to get an order and all its details, it better to store order as an aggregate object but dealing with these aggregates to get item details on all the orders is not elegant.

Aggregate-oriented databases make inter-aggregate relationships more difficult to handle than intra-aggregate relationships. Aggregate-ignorant databases are better when interactions use data organized in many different formations. Aggregate-oriented databases often compute materialized views to provide data organized differently from their primary aggregates. This is often done with map-reduce computations, such as a map-reduce job to get items sold per day.

KEY-VALUE AND DOCUMENT DATA MODELS

NoSQL databases are known for their flexibility and performance, but no single type of NoSQL database is right for all use cases.

Simplicity: Key-Value Databases

Key-value databases and document databases are quite similar. Key-value databases are the simplest of the NoSQL databases: The basic data structure is a dictionary or map. You can store a value, such as an integer, string, a JSON structure, or an array, along with a key used to reference that value. For example, a simple key-value database might have a value such as "Douglas Adams". This value is then assigned an ID, such as cust1237.

Using a JSON structure adds complexity to the database. For example, the database could store a full mailing address in addition to a person's name. In the previous example, key cust1237 could point to the following information:

```
{name: "Douglas Adams",  
  
street: "782 Southwest St.",  
  
city: "Austin",  
  
state: "TX"}
```

Key management is vital to smooth operations in a key-value database. Because key-value databases have no SQL-style query language to describe which values to fetch, keys are used to reference data. Some key-value databases compensate for the lack of a query language by incorporating search capabilities. Instead of searching by key, users can search values for particular patterns; for example, all values with a certain string in the name, such as "Douglas". However, freeform search is not available in all key-value databases.

Flexibility: Document Databases

Document databases such as [MongoDB](#) and [Couchbase](#) extend the concept of the key-value database. In fact, document databases maintain sets of key-value pairs within a document. The JSON example shown earlier is also a document. The term *document* in NoSQL databases refers to a set of key-value pairs, typically represented in JSON, XML, or a binary form of JSON.

Feature Comparison

If a document database is essentially a key-value database with more features, shouldn't you choose the option with more features and be done with it? Not necessarily.

Document databases organize documents into groups called *collections*, which are analogous to the tables in relational databases. By contrast, key-value databases store all key-value pairs together in a single namespace, which is analogous to a relational schema.

Key-value pairs of similar types, such as IDs and names, are stored with dissimilar value pairs, such as IDs and customer orders. This sounds like a potential problem, but whether it is depends on the search capabilities of the key-value database. When searching using freeform text to find the ID associated with a given name, you can search across all value types—not just names. However, unless search queries are properly crafted, this approach could end up being less efficient than using a document database.

Document database collections allow developers to apply a high level of organization to their databases. For example, a document database for an e-commerce site might include collections for customers, orders, and products:

- *Customers* would include fields such as name, shipping address, and billing address.
- *Orders* would include fields such as product and shipping address data.
- *Products* would include fields such as department and price data.

Document databases can also provide better performance when working with complex data sets.

Separating collection entities by type, such as orders and customer profiles, can help with performance. Similarly, large collections, such as a collection of products, can be partitioned to improve query performance. Partitioning splits collections over multiple servers, allocating a subset of work to each server.

Document databases also support indexing, which can improve query performance by using filter criteria, such as searching for all orders placed in the last 10 days. However, it's often best to limit the use of indexes, confining them to fields that are commonly used in filtering query results. Having too many indexes can slow write operations. In the end, using indexes is a

tradeoff. You may get faster query responses at the expense of slower write operations, plus the cost of additional space to store index data.

RELATIONSHIPS:

Aggregates are useful in that they put together data that is commonly accessed together. But there are still lots of cases where data that's related is accessed differently. Consider the relationship between a customer and all of his orders. Some applications will want to access the order history whenever they access the customer; this fits in well with combining the customer with his order history into a single aggregate. Other applications, however, want to process orders individually and thus model orders as independent aggregates.

In this case, you'll want separate order and customer aggregates but with some kind of relationship between them so that any work on an order can look up customer data. The simplest way to provide such a link is to embed the ID of the customer within the order's aggregate data. That way, if you need data from the customer record, you read the order, ferret out the customer ID, and make another call to the database to read the customer data.

This will work, and will be just fine in many scenarios—but the database will be ignorant of the relationship in the data. This can be important because there are times when it's useful for the database to know about these links. As a result, many databases—even key-value stores—provide ways to make these relationships visible to the database. Document stores make the content of the aggregate available to the database to form indexes and queries.

Riak, a key-value store, allows you to put link information in metadata, supporting partial retrieval and link-walking capability. An important aspect of relationships between aggregates is how they handle updates. Aggregate oriented databases treat the aggregate as the unit of data-retrieval. Consequently, atomicity is only supported within the contents of a single aggregate. If you update multiple aggregates at once, you have to deal yourself with a failure partway through. Relational databases help you with this by allowing you to modify multiple records in a single transaction, providing ACID guarantees while altering many rows.

All of this means that aggregate-oriented databases become more awkward as you need to operate across multiple aggregates. There are various ways to deal with this, which we'll explore later in this chapter, but the fundamental awkwardness remains. This may imply that if you have data based on lots of relationships, you should prefer a relational database over a NoSQL store. While that's true for aggregate-oriented databases, it's worth remembering that relational databases aren't all that stellar with complex relationships either. While you can express queries

involving joins in SQL, things quickly get very hairy—both with SQL writing and with the resulting performance—as the number of joins mounts up. This makes it a good moment to introduce another category of databases that’s often lumped into the NoSQL pile.

SCHEMALESS DATABASE:

A common theme across all the forms of NoSQL databases is that they are schemaless. When you want to store data in a relational database, you first have to define a schema—a defined structure for the database which says what tables exist, which columns exist, and what data types each column can hold. Before you store some data, you have to have the schema defined for it. With NoSQL databases, storing data is much more casual.

A key-value store allows you to store any data you like under a key. A document database effectively does the same thing, since it makes no restrictions on the structure of the documents you store. Column-family databases allow you to store any data under any column you like. Graph databases allow you to freely add new edges and freely add properties to nodes and edges as you wish. Advocates of schemalessness rejoice in this freedom and flexibility.

With a schema, you have to figure out in advance what you need to store, but that can be hard to do. Without a schema binding you, you can easily store whatever you need. This allows you to easily change your data storage as you learn more about your project. You can easily add new things as you discover them. Furthermore, if you find you don’t need some things anymore, you can just stop storing them, without worrying about losing old data as you would if you delete columns in a relational schema.

As well as handling changes, a schemaless store also makes it easier to deal with nonuniform data: data where each record has a different set of fields. A schema puts all rows of a table into a straightjacket, which becomes awkward if you have different kinds of data in different rows. You either end up with lots of columns that are usually null (a sparse table), or you end up with meaningless columns like custom column 4. Schemalessness avoids this, allowing each record to contain just what it needs—no more, no less.

Schemalessness is appealing, and it certainly avoids many problems that exist with fixed-schema databases, but it brings some problems of its own. If all you are doing is storing some data and displaying it in a report as a simple list of fieldName: value lines then a schema is only going to get in the way. But usually we do with our data more than this, and we do it with programs that need to know that the billing address is called billingAddress and not addressForBilling and that the quantify field is going to be an integer 5 and not five.

The vital, if sometimes inconvenient, fact is that whenever we write a program that accesses data, that program almost always relies on some form of implicit schema. Unless it just says something like Click here to view code image //pseudo code foreach (Record r in records) { foreach (Field f in r.fields) { print (f.name, f.value) } } it will assume that certain field names are present and carry data with a certain meaning, and assume something about the type of data stored within that field. Programs are not humans; they cannot read “qty” and infer that that must be the same as “quantity”—at least not unless we specifically program them to do so. So, however schemaless our database is, there is usually an implicit schema present. This implicit schema is a set of assumptions about the data’s structure in the code that manipulates the data.

Having the implicit schema in the application code results in some problems. It means that in order to understand what data is present you have to dig into the application code. If that code is well structured you should be able to find a clear place from which to deduce the schema. But there are no guarantees; it all depends on how clear the application code is. Furthermore, the database remains ignorant of the schema—it can’t use the schema to help it decide how to store and retrieve data efficiently. It can’t apply its own validations upon that data to ensure that different applications don’t manipulate data in an inconsistent way. These are the reasons why relational databases have a fixed schema, and indeed the reasons why most databases have had fixed schemas in the past. Schemas have value, and the rejection of schemas by NoSQL databases is indeed quite startling.

Essentially, a schemaless database shifts the schema into the application code that accesses it. This becomes problematic if multiple applications, developed by different people, access the same database. These problems can be reduced with a couple of approaches. One is to encapsulate all database interaction within a single application and integrate it with other applications using web services. This fits in well with many people’s current preference for using web services for integration. Another approach is to clearly delineate different areas of an aggregate for access by different applications. These could be different sections in a document database or different column families in a column-family database.

Although NoSQL fans often criticize relational schemas for having to be defined up front and being inflexible, that’s not really true. Relational schemas can be changed at any time with standard SQL commands. If necessary, you can create new columns in an ad-hoc way to store nonuniform data. We have only rarely seen this done, but it worked reasonably well where we have. Most of the time, however, nonuniformity in your data is a good reason to favor a schemaless database. Schemalessness does have a big impact on changes of a database’s structure over time, particularly for more uniform data. Although it’s not practiced as widely as it ought to be, changing a relational database’s schema can be done in a controlled way.

Similarly, you have to exercise control when changing how you store data in a schemaless database so that you can easily access both old and new data. Furthermore, the flexibility that schemalessness gives you only applies within an aggregate—if you need to change your aggregate boundaries, the migration is every bit as complex as it is in the relational case

MATERIALIZED VIEW:

When we talked about aggregate-oriented data models, we stressed their advantages. If you want to access orders, it's useful to have all the data for an order contained in a single aggregate that can be stored and accessed as a unit. But aggregate-orientation has a corresponding disadvantage: What happens if a product manager wants to know how much a particular item has sold over the last couple of weeks? Now the aggregate-orientation works against you, forcing you to potentially read every order in the database to answer the question.

You can reduce this burden by building an index on the product, but you're still working against the aggregate structure. Relational databases have an advantage here because their lack of aggregate structure allows them to support accessing data in different ways. Furthermore, they provide a convenient mechanism that allows you to look at data differently from the way it's stored—views.

A view is like a relational table (it is a relation) but it's defined by computation over the base tables. When you access a view, the database computes the data in the view—a handy form of encapsulation. Views provide a mechanism to hide from the client whether data is derived data or base data—but can't avoid the fact that some views are expensive to compute. To cope with this, materialized views were invented, which are views that are computed in advance and cached on disk. Materialized views are effective for data that is read heavily but can stand being somewhat stale.

Although NoSQL databases don't have views, they may have precomputed and cached queries, and they reuse the term “materialized view” to describe them. It's also much more of a central aspect for aggregate-oriented databases than it is for relational systems, since most applications will have to deal with some queries that don't fit well with the aggregate structure. There are two rough strategies to building a materialized view. The first is the eager approach where you update the materialized view at the same time you update the base data for it. In this case, adding an order would also update the purchase history aggregates for each product.

This approach is good when you have more frequent reads of the materialized view than you have writes and you want the materialized views to be as fresh as possible. The application database (p. 7) approach is valuable here as it makes it easier to ensure that any updates to base

data also update materialized views. If you don't want to pay that overhead on each update, you can run batch jobs to update the materialized views at regular intervals. You'll need to understand your business requirements to assess how stale your materialized views can be. You can build materialized views outside of the database by reading the data, computing the view, and saving it back to the database. More often databases will support building materialized views themselves.

In this case, you provide the computation that needs to be done, and the database executes the computation when needed according to some parameters that you configure. This is particularly handy for eager updates of views with incremental map-reduce. Materialized views can be used within the same aggregate. An order document might include an order summary element that provides summary information about the order so that a query for an order summary does not have to transfer the entire order document. Using different column families for materialized views is a common feature of column-family databases. An advantage of doing this is that it allows you to update the materialized view within the same atomic operation.

DISTRIBUTION MODELS:

The primary driver of interest in NoSQL has been its ability to run databases on a large cluster. As data volumes increase, it becomes more difficult and expensive to scale up—buy a bigger server to run the database on. A more appealing option is to scale out—run the database on a cluster of servers. Aggregate orientation fits well with scaling out because the aggregate is a natural unit to use for distribution. Depending on your distribution model, you can get a data store that will give you the ability to handle larger quantities of data, the ability to process a greater read or write traffic, or more availability in the face of network slowdowns or breakages. These are often important benefits, but they come at a cost. Running over a cluster introduces complexity—so it's not something to do unless the benefits are compelling. Broadly, there are two paths to data distribution: replication and sharding. Replication takes the same data and copies it over multiple nodes. Sharding puts different data on different nodes. Replication and sharding are orthogonal techniques: You can use either or both of them. Replication comes into two forms: master-slave and peer-to-peer. We will now discuss these techniques starting at the simplest and working up to the more complex: first single-server, then master-slave replication, then sharding, and finally peer-to-peer replication.

PEER-PEER REPLICATION

Master-slave replication helps with read scalability but doesn't help with scalability of writes. It provides resilience against failure of a slave, but not of a master. Essentially, the master is still a

bottleneck and a single point of failure. Peer-to-peer replication attacks these problems by not having a master. All the replicas have equal weight, they can all accept writes, and the loss of any of them doesn't prevent access to the data store.

The prospect here looks mighty fine. With a peer-to-peer replication cluster, you can ride over node failures without losing access to data. Furthermore, you can easily add nodes to improve your performance. There's much to like here—but there are complications. The biggest complication is, again, consistency. When you can write to two different places, you run the risk that two people will attempt to update the same record at the same time—a write-write conflict. Inconsistencies on read lead to problems but at least they are relatively transient. Inconsistent writes are forever. We'll talk more about how to deal with write inconsistencies later on, but for the moment we'll note a couple of broad options. At one end, we can ensure that whenever we write data, the replicas coordinate to ensure we avoid a conflict. This can give us just as strong a guarantee as a master, albeit at the cost of network traffic to coordinate the writes.

We don't need all the replicas to agree on the write, just a majority, so we can still survive losing a minority of the replica nodes. At the other extreme, we can decide to cope with an inconsistent write. There are contexts when we can come up with policy to merge inconsistent writes. In this case we can get the full performance benefit of writing to any replica. These points are at the ends of a spectrum where we trade off consistency for availability.

CONSISTENCY:

One of the biggest changes from a centralized relational database to a cluster-oriented NoSQL database is in how you think about consistency. Relational databases try to exhibit strong consistency by avoiding all the various inconsistencies that we'll shortly be discussing. Once you start looking at the NoSQL world, phrases such as "CAP theorem" and "eventual consistency" appear, and as soon as you start building something you have to think about what sort of consistency you need for your system. Consistency comes in various forms, and that one word covers a myriad of ways errors can creep into your life. So we're going to begin by talking about the various shapes consistency can take. After that we'll discuss why you may want to relax consistency (and its big sister, durability).

RELAXING CONSISTENCY

Consistency is a Good Thing—but, sadly, sometimes we have to sacrifice it. It is always possible to design a system to avoid inconsistencies, but often impossible to do so without making unbearable sacrifices in other characteristics of the system. As a result, we often have to tradeoff consistency for something else. While some architects see this as a disaster, we see it as part of

the inevitable tradeoffs involved in system design. Furthermore, different domains have different tolerances for inconsistency, and we need to take this tolerance into account as we make our decisions.

Trading off consistency is a familiar concept even in single-server relational database systems. Here, our principal tool to enforce consistency is the transaction, and transactions can provide strong consistency guarantees. However, transaction systems usually come with the ability to relax isolation levels, allowing queries to read data that hasn't been committed yet, and in practice we see most applications relax consistency down from the highest isolation level (serialized) in order to get effective performance. We most commonly see people using the read-committed transaction level, which eliminates some read-write conflicts but allows others.

Many systems forgo transactions entirely because the performance impact of transactions is too high. We've seen this in a couple different ways. On a small scale, we saw the popularity of MySQL during the days when it didn't support transactions. Many websites liked the high speed of MySQL and were prepared to live without transactions. At the other end of the scale, some very large websites, such as eBay [Pritchett], have had to forgo transactions in order to perform acceptably—this is particularly true when you need to introduce sharding. Even without these constraints, many application builders need to interact with remote systems that can't be properly included within a transaction boundary, so updating outside of transactions is a quite common occurrence for enterprise applications.

The CAP Theorem

In the NoSQL world it's common to refer to the CAP theorem as the reason why you may need to relax consistency. It was originally proposed by Eric Brewer in 2000 [Brewer] and given a formal proof by Seth Gilbert and Nancy Lynch [Lynch and Gilbert] a couple of years later. (You may also hear this referred to as Brewer's Conjecture.) The basic statement of the CAP theorem is that, given the three properties of Consistency, Availability, and Partition tolerance, you can only get two. Obviously this depends very much on how you define these three properties, and differing opinions have led to several debates on what the real consequences of the CAP theorem are. Consistency is pretty much as we've defined it so far. Availability has a particular meaning in the context of CAP—it means that if you can talk to a node in the cluster, it can read and write data. That's subtly different from the usual meaning, which we'll explore later. Partition tolerance means that the cluster can survive communication breakages in the cluster that separate the cluster into multiple partitions unable to communicate with each other.

A single-server system is the obvious example of a CA system—a system that has Consistency and Availability but not Partition tolerance. A single machine can't partition, so it does not have to worry about partition tolerance. There's only one node—so if it's up, it's available. Being up and keeping consistency is reasonable. This is the world that most relational database systems live in. It is theoretically possible to have a CA cluster.

However, this would mean that if a partition ever occurs in the cluster, all the nodes in the cluster would go down so that no client can talk to a node. By the usual definition of “available,” this would mean a lack of availability, but this is where CAP's special usage of “availability” gets confusing. CAP defines “availability” to mean “every request received by a nonfailing node in the system must result in a response” [Lynch and Gilbert]. So a failed, unresponsive node doesn't infer a lack of CAP availability. This does imply that you can build a CA cluster, but you have to ensure it will only partition rarely and completely.

This can be done, at least within a data center, but it's usually prohibitively expensive. Remember that in order to bring down all the nodes in a cluster on a partition, you also have to detect the partition in a timely manner—which itself is no small feat. So clusters have to be tolerant of network partitions. And here is the real point of the CAP theorem. Although the CAP theorem is often stated as “you can only get two out of three,” in practice what it's saying is that in a system that may suffer partitions, as distributed system do, you have to trade off consistency versus availability. This isn't a binary decision; often, you can trade off a little consistency to get some availability. The resulting system would be neither perfectly consistent nor perfectly available—but would have a combination that is reasonable for your particular needs.

An example should help illustrate this. Martin and Pramod are both trying to book the last hotel room on a system that uses peer-to-peer distribution with two nodes (London for Martin and Mumbai for Pramod). If we want to ensure consistency, then when Martin tries to book his room on the London node, that node must communicate with the Mumbai node before confirming the booking. Essentially, both nodes must agree on the serialization of their requests.

This gives us consistency—but should the network link break, then neither system can book any hotel room, sacrificing availability. One way to improve availability is to designate one node as the master for a particular hotel and ensure all bookings are processed by that master. Should that master be Mumbai, then Mumbai can still process hotel bookings for that hotel and Pramod will get the last room. If we use master-slave replication, London users can see the inconsistent room information but cannot make a booking and thus cause an update inconsistency.

However, users expect that it could happen in this situation—so, again, the compromise works for this particular use case. This improves the situation, but we still can't book a room on the London node for the hotel whose master is in Mumbai if the connection goes down. In CAP terminology, this is a failure of availability in that Martin can talk to the London node but the London node cannot update the data. To gain more availability, we might allow both systems to keep accepting hotel reservations even if the network link breaks down. The danger here is that Martin and Pramod book the last hotel room. However, depending on how this hotel operates, that may be fine. Often, travel companies tolerate a certain amount of overbooking in order to cope with no-shows. Conversely, some hotels always keep a few rooms clear even when they are fully booked, in order to be able to swap a guest out of a room with problems or to accommodate a high-status late booking.

Some might even cancel the booking with an apology once they detected the conflict—reasoning that the cost of that is less than the cost of losing bookings on network failures. The classic example of allowing inconsistent writes is the shopping cart, as discussed in Dynamo [Amazon's Dynamo]. In this case you are always allowed to write to your shopping cart, even if network failures mean you end up with multiple shopping carts. The checkout process can merge the two shopping carts by putting the union of the items from the carts into a single cart and returning that. Almost always that's the correct answer—but if not, the user gets the opportunity to look at the cart before completing the order. The lesson here is that although most software developers treat update consistency as The Way Things Must Be, there are cases where you can deal gracefully with inconsistent answers to requests.

These situations are closely tied to the domain and require domain knowledge to know how to resolve. Thus you can't usually look to solve them purely within the development team—you have to talk to domain experts. If you can find a way to handle inconsistent updates, this gives you more options to increase availability and performance. For a shopping cart, it means that shoppers can always shop, and do so quickly. And as Patriotic Americans, we know how vital it is to support Our Retail Destiny.

A similar logic applies to read consistency. If you are trading financial instruments over a computerized exchange, you may not be able to tolerate any data that isn't right up to date. However if you are posting a news item to a media website, you may be able to tolerate old pages for minutes. In these cases you need to know how tolerant you are of stale reads, and how long the inconsistency window can be—often in terms of the average length, worst case, and some measure of the distribution for the lengths. Different data items may have different tolerances for staleness, and thus may need different settings in your replication configuration. Advocates of NoSQL often say that instead of following the ACID properties of relational

transactions, NoSQL systems follow the BASE properties (Basically Available, Soft state, Eventual consistency) [Brewer].

Although we feel we ought to mention the BASE acronym here, we don't think it's very useful. The acronym is even more contrived than ACID, and neither "basically available" nor "soft state" have been well defined. We should also stress that when Brewer introduced the notion of BASE, he saw the tradeoff between ACID and BASE as a spectrum, not a binary choice. We've included this discussion of the CAP theorem because it's often used (and abused) when talking about the tradeoffs involving consistency in distributed databases.

However, it's usually better to think not about the tradeoff between consistency and availability but rather between consistency and latency. We can summarize much of the discussion about consistency in distribution by saying that we can improve consistency by getting more nodes involved in the interaction, but each node we add increases the response time of that interaction. We can then think of availability as the limit of latency that we're prepared to tolerate; once latency gets too high, we give up and treat the data as unavailable—which neatly fits its definition in the context of CAP.

VERSION STAMPS

Many critics of NoSQL databases focus on the lack of support for transactions. Transactions are a useful tool that helps programmers support consistency. One reason why many NoSQL proponents worry less about a lack of transactions is that aggregate-oriented NoSQL databases do support atomic updates within an aggregate—and aggregates are designed so that their data forms a natural unit of update. That said, it's true that transactional needs are something to take into account when you decide what database to use. As part of this, it's important to remember that transactions have limitations. Even within a transactional system we still have to deal with updates that require human intervention and usually cannot be run within transactions because they would involve holding a transaction open for too long. We can cope with these using version stamps—which turn out to be handy in other situations as well, particularly as we move away from the single-server distribution model

Business and System Transactions

The need to support update consistency without transactions is actually a common feature of systems even when they are built on top of transactional databases. When users think about transactions, they usually mean business transactions. A business transaction may be something like browsing a product catalog, choosing a bottle of Talisker at a good price, filling in credit card information, and confirming the order. Yet all of this usually won't occur within the system

transaction provided by the database because this would mean locking the database elements while the user is trying to find their credit card and gets called off to lunch by their colleagues. Usually applications only begin a system transaction at the end of the interaction with the user, so that the locks are only held for a short period of time.

The problem, however, is that calculations and decisions may have been made based on data that's changed. The price list may have updated the price of the Talisker, or someone may have updated the customer's address, changing the shipping charges. The broad techniques for handling this are offline concurrency [Fowler PoEAA], useful in NoSQL situations too. A particularly useful approach is the Optimistic Offline Lock [Fowler PoEAA], a form of conditional update where a client operation rereads any information that the business transaction relies on and checks that it hasn't changed since it was originally read and displayed to the user. A good way of doing this is to ensure that records in the database contain some form of version stamp: a field that changes every time the underlying data in the record changes.

When you read the data you keep a note of the version stamp, so that when you write data you can check to see if the version has changed. You may have come across this technique with updating resources with HTTP [HTTP]. One way of doing this is to use etags. Whenever you get a resource, the server responds with an etag in the header. This etag is an opaque string that indicates the version of the resource. If you then update that resource, you can use a conditional update by supplying the etag that you got from your last GET. If the resource has changed on the server, the etags won't match and the server will refuse the update, returning a 412 (Precondition Failed) response. Some databases provide a similar mechanism of conditional update that allows you to ensure updates won't be based on stale data. You can do this check yourself, although you then have to ensure no other thread can run against the resource between your read and your update. (Sometimes this is called a compare-and-set (CAS) operation, whose name comes from the CAS operations done in processors.

The difference is that a processor CAS compares a value before setting it, while a database conditional update compares a version stamp of the value.) There are various ways you can construct your version stamps. You can use a counter, always incrementing it when you update the resource. Counters are useful since they make it easy to tell if one version is more recent than another. On the other hand, they require the server to generate the counter value, and also need a single master to ensure the counters aren't duplicated.

Another approach is to create a GUID, a large random number that's guaranteed to be unique. These use some combination of dates, hardware information, and whatever other sources of randomness they can pick up. The nice thing about GUIDs is that they can be generated by

anyone and you'll never get a duplicate; a disadvantage is that they are large and can't be compared directly for recentness. A third approach is to make a hash of the contents of the resource. With a big enough hash key size, a content hash can be globally unique like a GUID and can also be generated by anyone; the advantage is that they are deterministic—any node will generate the same content hash for same resource data. However, like GUIDs they can't be directly compared for recentness, and they can be lengthy.

A fourth approach is to use the timestamp of the last update. Like counters, they are reasonably short and can be directly compared for recentness, yet have the advantage of not needing a single master. Multiple machines can generate timestamps—but to work properly, their clocks have to be kept in sync. One node with a bad clock can cause all sorts of data corruptions. There's also a danger that if the timestamp is too granular you can get duplicates—it's no good using timestamps of a millisecond precision if you get many updates per millisecond. You can blend the advantages of these different version stamp schemes by using more than one of them to create a composite stamp. For example, CouchDB uses a combination of counter and content hash. Most of the time this allows version stamps to be compared for recentness, even when you use peer-to-peer replication. Should two peers update at the same time, the combination of the same count and different content hashes makes it easy to spot the conflict. As well as helping to avoid update conflicts, version stamps are also useful for providing session consistency.

Version Stamps on Multiple Nodes

The basic version stamp works well when you have a single authoritative source for data, such as a single server or master-slave replication. In that case the version stamp is controlled by the master. Any slaves follow the master's stamps. But this system has to be enhanced in a peer-to-peer distribution model because there's no longer a single place to set the version stamps. If you're asking two nodes for some data, you run into the chance that they may give you different answers. If this happens, your reaction may vary depending on the cause of that difference. It may be that an update has only reached one node but not the other, in which case you can accept the latest (assuming you can tell which one that is). Alternatively, you may have run into an inconsistent update, in which case you need to decide how to deal with that. In this situation, a simple GUID or etag won't suffice, since these don't tell you enough about the relationships.

The simplest form of version stamp is a counter. Each time a node updates the data, it increments the counter and puts the value of the counter into the version stamp. If you have blue and green slave replicas of a single master, and the blue node answers with a version stamp of 4 and the green node with 6, you know that the green's answer is more recent. In multiple-master cases,

we need something fancier. One approach, used by distributed version control systems, is to ensure that all nodes contain a history of version stamps.

That way you can see if the blue node's answer is an ancestor of the green's answer. This would either require the clients to hold onto version stamp histories, or the server nodes to keep version stamp histories and include them when asked for data. This also detects an inconsistency, which we would see if we get two version stamps and neither of them has the other in their histories. Although version control systems keep these kinds of histories, they aren't found in NoSQL databases. A simple but problematic approach is to use timestamps.

The main problem here is that it's usually difficult to ensure that all the nodes have a consistent notion of time, particularly if updates can happen rapidly. Should a node's clock get out of sync, it can cause all sorts of trouble. In addition, you can't detect write-write conflicts with timestamps, so it would only work well for the singlemaster case—and then a counter is usually better. The most common approach used by peer-to-peer NoSQL systems is a special form of version stamp which we call a vector stamp. In essence, a vector stamp is a set of counters, one for each node.

A vector stamp for three nodes (blue, green, black) would look something like [blue: 43, green: 54, black: 12]. Each time a node has an internal update, it updates its own counter, so an update in the green node would change the vector to [blue: 43, green: 55, black: 12]. Whenever two nodes communicate, they synchronize their vector stamps. There are several variations of exactly how this synchronization is done.

We're coining the term "vector stamp" as a general term in this book; you'll also come across vector clocks and version vectors—these are specific forms of vector stamps that differ in how they synchronize. By using this scheme you can tell if one version stamp is newer than another because the newer stamp will have all its counters greater than or equal to those in the older stamp.

So [blue: 1, green: 2, black: 5] is newer than [blue:1, green: 1, black 5] since one of its counters is greater. If both stamps have a counter greater than the other, e.g. [blue: 1, green: 2, black: 5] and [blue: 2, green: 1, black: 5], then you have a write-write conflict. There may be missing values in the vector, in which case we use treat the missing value as 0. So [blue: 6, black: 2] would be treated as [blue: 6, green: 0, black: 2]. This allows you to easily add new nodes without invalidating the existing vector stamps. Vector stamps are a valuable tool that spots inconsistencies, but doesn't resolve them. Any conflict resolution will depend on the domain you are working in. This is part of the consistency/latency tradeoff. You either have to live with the

fact that network partitions may make your system unavailable, or you have to detect and deal with inconsistencies.

PARTITIONING AND COMBINING

In the simplest form, we think of a map-reduce job as having a single reduce function. The outputs from all the map tasks running on the various nodes are concatenated together and sent into the reduce. While this will work, there are things we can do to increase the parallelism and to reduce the data transfer. The first thing we can do is increase parallelism by partitioning the output of the mappers. Each reduce function operates on the results of a single key. This is a limitation—it means you can't do anything in the reduce that operates across keys—but it's also a benefit in that it allows you to run multiple reducers in parallel.

To take advantage of this, the results of the mapper are divided up based the key on each processing node. Typically, multiple keys are grouped together into partitions. The framework then takes the data from all the nodes for one partition, combines it into a single group for that partition, and sends it off to a reducer. Multiple reducers can then operate on the partitions in parallel, with the final results merged together. (This step is also called “shuffling,” and the partitions are sometimes referred to as “buckets” or “regions.”) The next problem we can deal with is the amount of data being moved from node to node between the map and reduce stages. Much of this data is repetitive, consisting of multiple key-value pairs for the same key. A combiner function cuts this data down by combining all the data for the same key into a single value.

A combiner function is, in essence, a reducer function—indeed, in many cases the same function can be used for combining as the final reduction. The reduce function needs a special shape for this to work: Its output must match its input. We call such a function a combinable reducer. Not all reduce functions are combinable. Consider a function that counts the number of unique customers for a particular product. The map function for such an operation would need to emit the product and the customer. The reducer can then combine them and count how many times each customer appears for a particular product, emitting the product and the count (see Figure 7.5). But this reducer's output is different from its input, so it can't be used as a combiner. You can still run a combining function here: one that just eliminates duplicate product-customer pairs, but it will be different from the final reducer. When you have combining reducers, the map-reduce framework can safely run not only in parallel (to reduce different partitions), but also in series to reduce the same partition at different times and places. In addition to allowing combining to occur on a node before data transmission, you can also start combining before mappers have finished.

This provides a good bit of extra flexibility to the map-reduce processing. Some map-reduce frameworks require all reducers to be combining reducers, which maximizes this flexibility. If you need to do a noncombining reducer with one of these frameworks, you'll need to separate the processing into pipelined map-reduce steps.

COMPOSING MAP-REDUCE CALCULATIONS

The map-reduce approach is a way of thinking about concurrent processing that trades off flexibility in how you structure your computation for a relatively straightforward model for parallelizing the computation over a cluster. Since it's a tradeoff, there are constraints on what you can do in your calculations. Within a map task, you can only operate on a single aggregate. Within a reduce task, you can only operate on a single key. This means you have to think differently about structuring your programs so they work well within these constraints. One simple limitation is that you have to structure your calculations around operations that fit in well with the notion of a reduce operation. A good example of this is calculating averages. Let's consider the kind of orders we've been looking at so far; suppose we want to know the average ordered quantity of each product. An important property of averages is that they are not composable—that is, if I take two groups of orders, I can't combine their averages alone. Instead, I need to take total amount and the count of orders from each group, combine those, and then calculate the average from the combined sum and count

This notion of looking for calculations that reduce neatly also affects how we do counts. To make a count, the mapping function will emit count fields with a value of 1, which can be summed to get a total count

A Two Stage Map-Reduce Example

As map-reduce calculations get more complex, it's useful to break them down into stages using a pipes-and-filters approach, with the output of one stage serving as input to the next, rather like the pipelines in UNIX. Consider an example where we want to compare the sales of products for each month in 2011 to the prior year. To do this, we'll break the calculations down into two stages. The first stage will produce records showing the aggregate figures for a single product in a single month of the year. The second stage then uses these as inputs and produces the result for a single product by comparing one month's results with the same month in the prior year

A first stage would read the original order records and output a series of key-value pairs for the sales of each product per month.

This stage is similar to the map-reduce examples we've seen so far. The only new feature is using a composite key so that we can reduce records based on the values of multiple fields. The second-stage mappers process this output depending on the year. A 2011 record populates the current year quantity while a 2010 record populates a prior year quantity. Records for earlier years (such as 2009) don't result in any mapping output being emitted.

The reduce in this case is a merge of records, where combining the values by summing allows two different year outputs to be reduced to a single value (with a calculation based on the reduced values thrown in for good measure)

Decomposing this report into multiple map-reduce steps makes it easier to write. Like many transformation examples, once you've found a transformation framework that makes it easy to compose steps, it's usually easier to compose many small steps together than try to cram heaps of logic into a single step. Another advantage is that the intermediate output may be useful for different outputs too, so you can get some reuse. This reuse is important as it saves time both in programming and in execution. The intermediate records can be saved in the data store, forming a materialized view.

Map-reduce is a pattern that can be implemented in any programming language. However, the constraints of the style make it a good fit for languages specifically designed for map-reduce computations. Apache Pig [Pig], an offshoot of the Hadoop [Hadoop] project, is a language specifically built to make it easy to write map-reduce programs. It certainly makes it much easier to work with Hadoop than the underlying Java libraries. In a similar vein, if you want to specify mapreduce programs using an SQL-like syntax, there is hive [Hive], another Hadoop offshoot. The map-reduce pattern is important to know about even outside of the context of NoSQL databases. Google's original map-reduce system operated on files stored on a distributed file system—an approach that's used by the open-source Hadoop project. While it takes some thought to get used to the constraints of structuring computations in map-reduce steps, the result is a calculation that is inherently well-suited to running on a cluster. When dealing with high volumes of data, you need to take a cluster-oriented approach. Aggregate-oriented databases fit well with this style of calculation We think that in the next few years many more organizations will be processing the volumes of data that demand a cluster-oriented solution—and the map-reduce pattern will see more and more use.

Incremental Map-Reduce

The examples we've discussed so far are complete map-reduce computations, where we start with raw inputs and create a final output. Many map-reduce computations take a while to

perform, even with clustered hardware, and new data keeps coming in which means we need to rerun the computation to keep the output up to date. Starting from scratch each time can take too long, so often it's useful to structure a map-reduce computation to allow incremental updates, so that only the minimum computation needs to be done. The map stages of a map-reduce are easy to handle incrementally—only if the input data changes does the mapper need to be rerun. Since maps are isolated from each other, incremental updates are straightforward. The more complex case is the reduce step, since it pulls together the outputs from many maps and any change in the map outputs could trigger a new reduction. This recomputation can be lessened depending on how parallel the reduce step is. If we are partitioning the data for reduction, then any partition that's unchanged does not need to be re-reduced. Similarly, if there's a combiner step, it doesn't need to be rerun if its source data hasn't changed.

If our reducer is combinable, there's some more opportunities for computation avoidance. If the changes are additive—that is, if we are only adding new records but are not changing or deleting any old records—then we can just run the reduce with the existing result and the new additions. If there are destructive changes, that is updates and deletes, then we can avoid some recomputation by breaking up the reduce operation into steps and only recalculating those steps whose inputs have changed—essentially, using a Dependency Network [Fowler DSL] to organize the computation. The map-reduce framework controls much of this, so you have to understand how a specific framework supports incremental operation.

DOCUMENT BASED DATABASE

Documents are the main concept in document databases. The database stores and retrieves documents, which can be XML, JSON, BSON, and so on. These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values. The documents stored are similar to each other but do not have to be exactly the same. Document databases store documents in the value part of the key-value store; think about document databases as key-value stores where the value is examinable. Let's look at how terminology compares in Oracle and MongoDB.

What Is a Document Database?

```
{ "firstname": "Martin",  
  
  "likes": [ "Biking",  
  
    "Photography" ],
```

```
"lastcity": "Boston",
```

```
"lastVisited":
```

```
}
```

The above document can be considered a row in a traditional RDBMS. Let's look at another document:

```
{
```

```
"firstname": "Pramod",
```

```
"citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],
```

```
"addresses": [
```

```
{ "state": "AK",
```

```
"city": "DILLINGHAM",
```

```
"type": "R"
```

```
},
```

```
{ "state": "MH",
```

```
"city": "PUNE",
```

```
"type": "R" }
```

```
],
```

```
"lastcity": "Chicago"
```

```
}
```

Looking at the documents, we can see that they are similar, but have differences in attribute names. This is allowed in document databases. The schema of the data can differ across documents, but these documents can still belong to the same collection—unlike an RDBMS where every row in a table has to follow the same schema. We represent a list of citiesvisited as an array, or a list of addresses as list of documents embedded inside the main document. Embedding child documents as subobjects inside documents provides for easy access and better

performance. If you look at the documents, you will see that some of the attributes are similar, such as firstname or city. At the same time, there are attributes in the second document which do not exist in the first document, such as addresses, while likes is in the first document but not the second. This different representation of data is not the same as in RDBMS where every column has to be defined, and if it does not have data it is marked as empty or set to null. In documents, there are no empty attributes; if a given attribute is not found, we assume that it was not set or not relevant to the document. Documents allow for new attributes to be created without the need to define them or to change the existing documents

Features

While there are many specialized document databases, we will use MongoDB as a representative of the feature set. Keep in mind that each product has some features that may not be found in other document databases. Let's take some time to understand how MongoDB works. Each MongoDB instance has multiple databases, and each database can have multiple collections. When we compare this with RDBMS, an RDBMS instance is the same as MongoDB instance, the schemas in RDBMS are similar to MongoDB databases, and the RDBMS tables are collections in MongoDB. When we store a document, we have to choose which database and collection this document belongs in—for example, `database.collection.insert(document)`, which is usually represented as `db.coll.insert(document)`.

Consistency

Consistency in MongoDB database is configured by using the replica sets and choosing to wait for the writes to be replicated to all the slaves or a given number of slaves. Every write can specify the number of servers the write has to be propagated to before it returns as successful. A command like `db.runCommand({ getlasterror : 1 , w : "majority" })` tells the database how strong is the consistency you want. For example, if you have one server and specify the w as majority, the write will return immediately since there is only one node. If you have three nodes in the replica set and specify w as majority, the write will have to complete at a minimum of two nodes before it is reported as a success. You can increase the w value for stronger consistency but you will suffer on write performance, since now the writes have to complete at more nodes. Replica sets also allow you to increase the read performance by allowing reading from slaves by setting `slaveOk`; this parameter can be set on the connection, or database, or collection, or individually for each operation

```
Mongo mongo = new Mongo("localhost:27017");
```

```
mongo.slaveOk();
```

Here we are setting slaveOk per operation, so that we can decide which operations can work with data from the slave node.

DBCollection collection = getOrderCollection(); BasicDBObject query = new BasicDBObject(); query.put("name", "Martin"); DBCursor cursor = collection.find(query).slaveOk(); Similar to various options available for read, you can change the settings to achieve strong write consistency, if desired. By default, a write is reported successful once the database receives it; you can change this so as to wait for the writes to be synced to disk or to propagate to two or more slaves. This is known as WriteConcern: You make sure that certain writes are written to the master and some slaves by setting WriteConcern to REPLICAS_SAFE. Shown below is code where we are setting the WriteConcern for all writes to a collection:

```
DBCollection shopping = database.getCollection("shopping");  
shopping.setWriteConcern(REPLICAS_SAFE);
```

WriteConcern can also be set per operation by specifying it on the save command:

WriteResult result = shopping.insert(order, REPLICAS_SAFE); There is a tradeoff that you need to carefully think about, based on your application needs and business requirements, to decide what settings make sense for slaveOk during read or what safety level you desire during write with WriteConcern.

Transactions

Transactions, in the traditional RDBMS sense, mean that you can start modifying the database with insert, update, or delete commands over different tables and then decide if you want to keep the changes or not by using commit or rollback. These constructs are generally not available in NoSQL solutions—a write either succeeds or fails. Transactions at the single-document level are known as atomic transactions. Transactions involving more than one operation are not possible, although there are products such as RavenDB that do support transactions across multiple operations. By default, all writes are reported as successful. A finer control over the write can be achieved by using WriteConcern parameter. We ensure that order is written to more than one node before it's reported successful by using WriteConcern.REPLICAS_SAFE. Different levels of WriteConcern let you choose the safety level during writes; for example, when writing log entries, you can use lowest level of safety, WriteConcern.NONE.

Availability

The CAP theorem ("The CAP Theorem," p. 53) dictates that we can have only two of Consistency, Availability, and Partition Tolerance. Document databases try to improve on

availability by replicating data using the master-slave setup. The same data is available on multiple nodes and the clients can get to the data even when the primary node is down. Usually, the application code does not have to determine if the primary node is available or not. MongoDB implements replication, providing high availability using replica sets. In a replica set, there are two or more nodes participating in an asynchronous master-slave replication. The replica-set nodes elect the master, or primary, among themselves. Assuming all the nodes have equal voting rights, some nodes can be favored for being closer to the other servers, for having more RAM, and so on; users can affect this by assigning a priority—a number between 0 and 1000—to a node. All requests go to the master node, and the data is replicated to the slave nodes. If the master node goes down, the remaining nodes in the replica set vote among themselves to elect a new master; all future requests are routed to the new master, and the slave nodes start getting data from the new master. When the node that failed comes back online, it joins in as a slave and catches up with the rest of the nodes by pulling all the data it needs to get current. Figure 9.1 is an example configuration of replica sets. We have two nodes, mongo A and mongo B, running the MongoDB database in the primary data-center, and mongo C in the secondary datacenter. If we want nodes in the primary datacenter to be elected as primary nodes, we can assign them a higher priority than the other nodes. More nodes can be added to the replica sets without having to take them offline.

Query Features

Document databases provide different query features. CouchDB allows you to query via views—complex queries on documents which can be either materialized (“Materialized Views,” p. 30) or dynamic (think of them as RDBMS views which are either materialized or not). With CouchDB, if you need to aggregate the number of reviews for a product as well as the average rating, you could add a view implemented via map-reduce (“Basic Map-Reduce,” p. 68) to return the count of reviews and the average of their ratings. When there are many requests, you don’t want to compute the count and average for every request; instead you can add a materialized view that precomputes the values and stores the results in the database. These materialized views are updated when queried, if any data was changed since the last update. One of the good features of document databases, as compared to key-value stores, is that we can query the data inside the document without having to retrieve the whole document by its key and then introspect the document. This feature brings these databases closer to the RDBMS query model. MongoDB has a query language which is expressed via JSON and has constructs such as \$query for the where clause, \$orderby for sorting the data, or \$explain to show the execution plan of the query. There are many more constructs like these that can be combined to create a MongoDB

query. Let's look at certain queries that we can do against MongoDB. Suppose we want to return all the documents in an order collection (all rows in the order table). The SQL for this would be:

SELECT * FROM order The equivalent query in Mongo shell would be: db.order.find()

Scaling

The idea of scaling is to add nodes or change data storage without simply migrating the database to a bigger box. We are not talking about making application changes to handle more load; instead, we are interested in what features are in the database so that it can handle more load. Scaling for heavy-read loads can be achieved by adding more read slaves, so that all the reads can be directed to the slaves. Given a heavy-read application, with our 3-node replica-set cluster, we can add more read capacity to the cluster as the read load increases just by adding more slave nodes to the replica set to execute reads with the slaveOk flag (Figure 9.2). This is horizontal scaling for reads.

Suitable Use Cases

Event Logging

Applications have different event logging needs; within the enterprise, there are many different applications that want to log events. Document databases can store all these different types of events and can act as a central data store for event storage. This is especially true when the type of data being captured by the events keeps changing. Events can be sharded by the name of the application where the event originated or by the type of event such as order_processed or customer_logged.

Content Management Systems, Blogging Platforms

Since document databases have no predefined schemas and usually understand JSON documents, they work well in content management systems or applications for publishing websites, managing user comments, user registrations, profiles, web-facing documents.

Web Analytics or Real-Time Analytics

Document databases can store data for real-time analytics; since parts of the document can be updated, it's very easy to store page views or unique visitors, and new metrics can be easily added without schema changes.

E-Commerce Applications

E-commerce applications often need to have flexible schema for products and orders, as well as the ability to evolve their data models without expensive database refactoring or data migration.

When Not to Use

There are problem spaces where document databases are not the best solution. 9.4.1. Complex Transactions Spanning Different Operations If you need to have atomic cross-document operations, then document databases may not be for you. However, there are some document databases that do support these kinds of operations, such as RavenDB. 9.4.2. Queries against Varying Aggregate Structure Flexible schema means that the database does not enforce any restrictions on the schema. Data is saved in the form of application entities. If you need to query these entities ad hoc, your queries will be changing (in RDBMS terms, this would mean that as you join criteria between tables, the tables to join keep changing). Since the data is saved as an aggregate, if the design of the aggregate is constantly changing, you need to save the aggregates at the lowest level of granularity—basically, you need to normalize the data. In this scenario, document databases may not work.

MONGO:

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)
Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

DATAMODEL:

Data in MongoDB has a flexible schema. documents in the same collection. They do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Some considerations while designing Schema in MongoDB

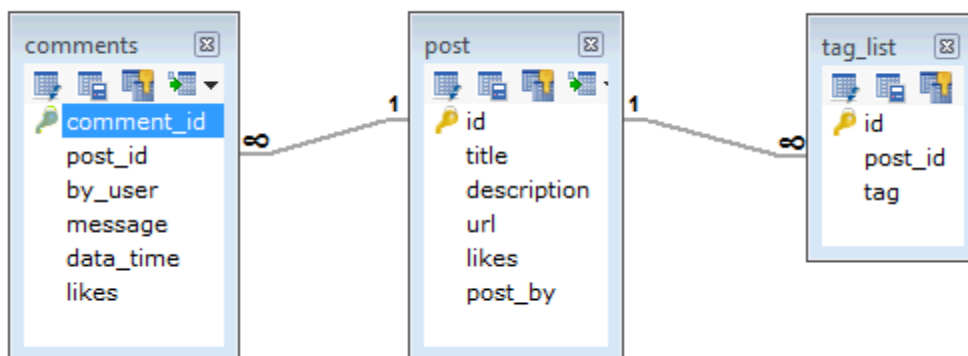
- Design your schema according to user requirements.
- Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
- Duplicate the data (but limited) because disk space is cheap as compare to compute time.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases.
- Do complex aggregation in the schema.

Example

Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.

- Every post has the unique title, description and url.
- Every post can have one or more tags.
- Every post has the name of its publisher and total number of likes.
- Every post has comments given by users along with their name, message, data-time and likes.
- On each post, there can be zero or more comments.

In RDBMS schema, design for above requirements will have minimum three tables.



While in MongoDB schema, design will have one collection post and the following structure –

```
{
  _id: POST_ID
  title: TITLE_OF_POST,
  description: POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    },
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    }
  ]
}
```

```
}
```

So while showing the data, in RDBMS you need to join three tables and in MongoDB, data will be shown from one collection only.

WORKING WITH DATA

The use Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows –

```
use DATABASE_NAME
```

Example

If you want to use a database with name **<mydb>**, then **use DATABASE** statement would be as follows –

```
>use mydb
```

```
switched to db mydb
```

To check your currently selected database, use the command **db**

```
>db  
mydb
```

If you want to check your databases list, use the command **show dbs**.

```
>show dbs  
local 0.78125GB  
test 0.23012GB
```

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

```
>db.movie.insert({"name":"tutorials point"})
```

```
>show dbs
```

```
local    0.78125GB
```

```
mydb     0.23012GB
```

```
test     0.23012GB
```

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax

Basic syntax of **dropDatabase()** command is as follows –

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Example

First, check the list of available databases by using the command, **show dbs**.

```
>show dbs
```

```
local    0.78125GB
```

```
mydb     0.23012GB
```

```
test     0.23012GB
```

```
>
```

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows –

```
>use mydb
```

```
switched to db mydb
```

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

Now check list of databases.

```
>show dbs  
  
local    0.78125GB  
test     0.23012GB  
  
>
```

REPLICATION:

Replication is the process of synchronizing data across multiple servers. Replication provides redundancy and increases data availability with multiple copies of data on different database servers. Replication protects a database from the loss of a single server. Replication also allows you to recover from hardware failure and service interruptions. With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

Why Replication?

- To keep your data safe
- High (24*7) availability of data
- Disaster recovery
- No downtime for maintenance (like backups, index rebuilds, compaction)
- Read scaling (extra copies to read from)
- Replica set is transparent to the application

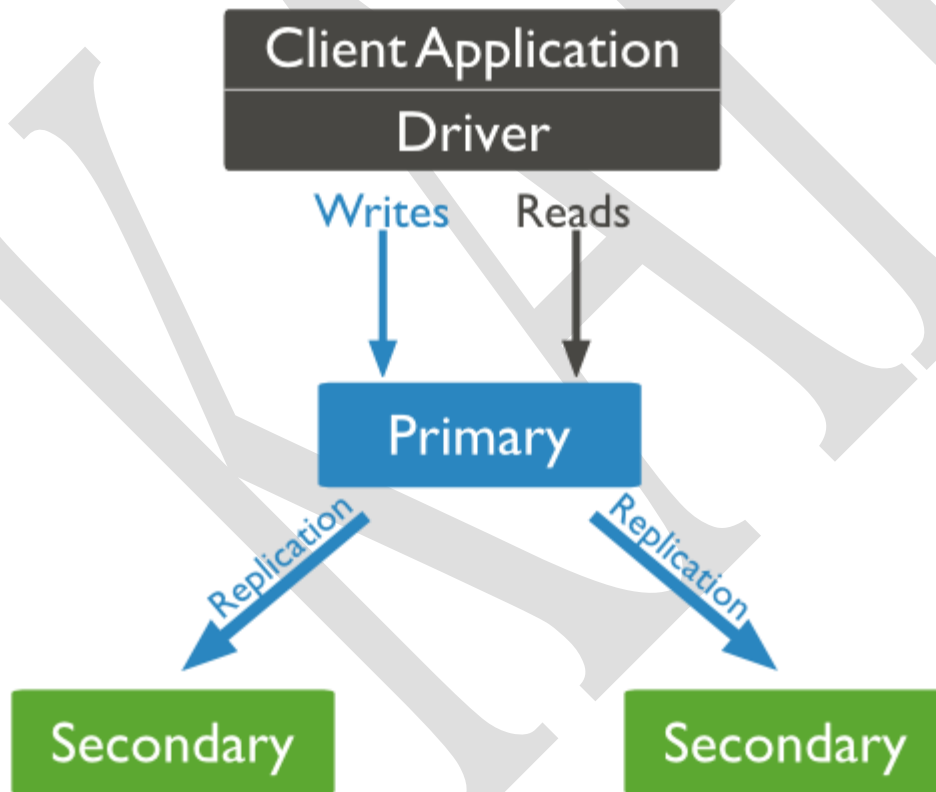
How Replication Works in MongoDB

MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set. In a replica, one node is primary node that

receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have the same data set. Replica set can have only one primary node.

- Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
- In a replica set, one node is primary node and remaining nodes are secondary.
- All data replicates from primary to secondary node.
- At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
- After the recovery of failed node, it again join the replica set and works as a secondary node.

A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.



- A cluster of N nodes
- Any one node can be primary
- All write operations go to primary
- Automatic failover
- Automatic recovery
- Consensus election of primary

Set Up a Replica Set

In this tutorial, we will convert standalone MongoDB instance to a replica set. To convert to replica set, following are the steps –

- Shutdown already running MongoDB server.
-
- Start the MongoDB server by specifying -- replSet option. Following is the basic syntax of --replSet –

```
mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet  
"REPLICA_SET_INSTANCE_NAME"
```

Example

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

- It will start a mongod instance with the name rs0, on port 27017.
- Now start the command prompt and connect to this mongod instance.
- In Mongo client, issue the command **rs.initiate()** to initiate a new replica set.
- To check the replica set configuration, issue the command **rs.conf()**. To check the status of replica set issue the command **rs.status()**.

Add Members to Replica Set

To add members to replica set, start mongod instances on multiple machines. Now start a mongo client and issue a command **rs.add()**.

Syntax

The basic syntax of **rs.add()** command is as follows –

```
>rs.add(HOST_NAME:PORT)
```

Example

Suppose your mongod instance name is **mongod1.net** and it is running on port **27017**. To add this instance to replica set, issue the command **rs.add()** in Mongo client.

```
>rs.add("mongod1.net:27017")
```

```
>
```

You can add mongod instance to replica set only when you are connected to primary node. To check whether you are connected to primary or not, issue the command **db.isMaster()** in mongo client.

SHARDING

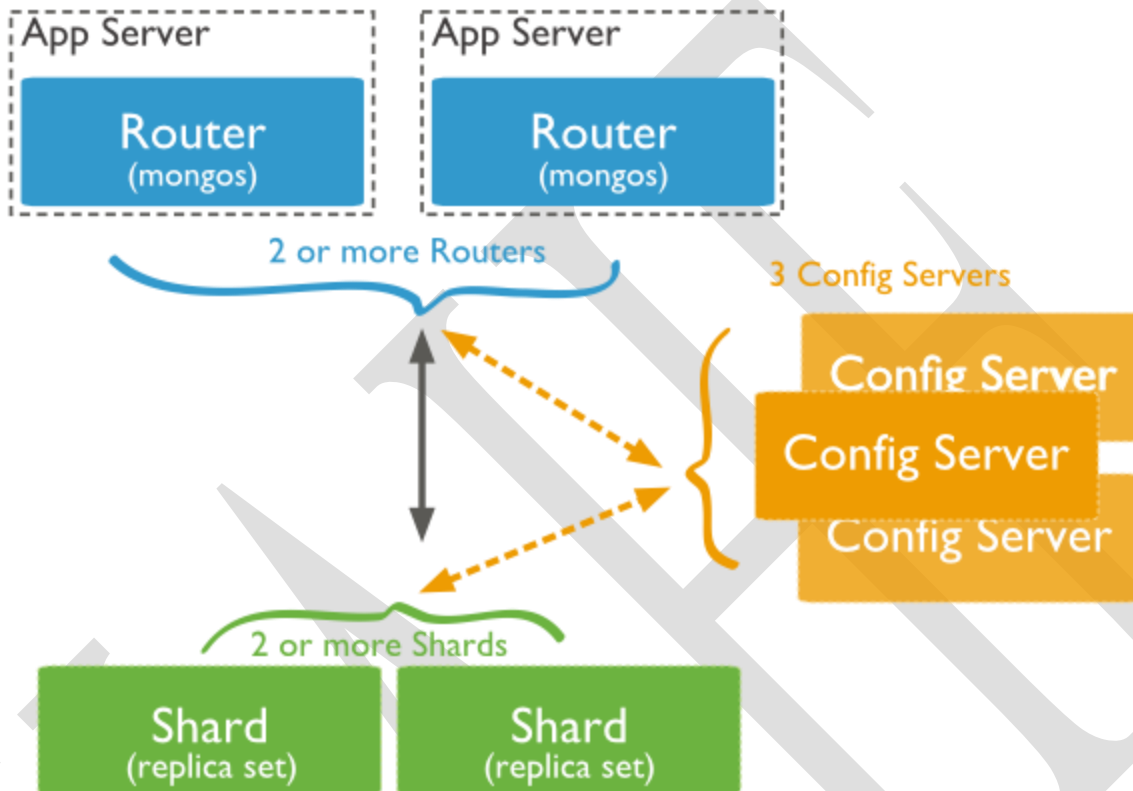
Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth. As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Why Sharding?

- In replication, all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes
- Memory can't be large enough when active dataset is big
- Local disk is not big enough
- Vertical scaling is too expensive

Sharding in MongoDB

The following diagram shows the sharding in MongoDB using sharded cluster.



In the following diagram, there are three main components –

- **Shards** – Shards are used to store data. They provide high availability and data consistency. In production environment, each shard is a separate replica set.
- **Config Servers** – Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment, sharded clusters have exactly 3 config servers.
- **Query Routers** – Query routers are basically mongo instances, interface with client applications and direct operations to the appropriate shard. The query router processes and targets the operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client

sends requests to one query router. Generally, a sharded cluster have many query routers.

DEPLOYMENT:

When you are preparing a MongoDB deployment, you should try to understand how your application is going to hold up in production. It's a good idea to develop a consistent, repeatable approach to managing your deployment environment so that you can minimize any surprises once you're in production.

The best approach incorporates prototyping your set up, conducting load testing, monitoring key metrics, and using that information to scale your set up. The key part of the approach is to proactively monitor your entire system - this will help you understand how your production system will hold up before deploying, and determine where you will need to add capacity. Having insight into potential spikes in your memory usage, for example, could help put out a write-lock fire before it starts.

To monitor your deployment, MongoDB provides some of the following commands –

`mongostat`

This command checks the status of all running mongod instances and return counters of database operations. These counters include inserts, queries, updates, deletes, and cursors. Command also shows when you're hitting page faults, and showcase your lock percentage. This means that you're running low on memory, hitting write capacity or have some performance issue.

`mongotop`

This command tracks and reports the read and write activity of MongoDB instance on a collection basis. By default, **mongotop** returns information in each second, which you can change it accordingly. You should check that this read and write activity matches your application intention, and you're not firing too many writes to the database at a time, reading too frequently from a disk, or are exceeding your working set size.

To run the command, start your mongod instance. In another command prompt, go to **bin** directory of your mongodb installation and type **mongotop**.

```
D:\set up\mongodb\bin>mongotop
```

To change **mongotop** command to return information less frequently, specify a specific number after the mongotop command.

```
D:\set up\mongodb\bin>mongotop 30
```

The above example will return values every 30 seconds.

Apart from the MongoDB tools, 10gen provides a free, hosted monitoring service, MongoDB Management Service (MMS), that provides a dashboard and gives you a view of the metrics from your entire cluster.

POSSIBLE QUESTIONS:

PART-B

(5 x 6=30)

1. Write a note on (i)Relationship (ii) Schemaless database
2. Write a note on (i)distribution models(s)(ii)peer-peer replication(s)
3. Explain about Replication &Sharding
4. Explain about Aggregates
5. Write a note on (i)consistency (ii)relaxing consistency
6. Explain about Document based Database
7. Describe about version stamps

PART-C

(1 x 10 = 10)

1. Explain about composing map-reduce calculations
2. Explain about Document based Database



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS) (BATCH 2018-2020)

BIG DATA ANALYTICS

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

ONE MARK QUESTIONS

	UNIT-4	choice 1	choice 2	choice 3	choice 4			Answer
1	_____ commodity Hardware in Hadoop world mean.	Very cheap hardware	Industry standard hardware	c) Discarded hardware	a) Low specifications Industry grade			Low specifications Industry grade hardware
2	_____ are NOT big data problem(s)	Parsing 5 MB XML file every 5 minutes	Processing IPL tweet sentiments and online	c) Processing online bank transactions	d) Parsing transactions			Processing IPL tweet sentiments and line bank
3	_____ “Velocity” in Big Data mean	Speed of input data generation	Speed of individual machine processors	c) Speed of ONLY storing data	a) Speed of storing and processing data			Speed of storing and processing data
4	The term Big Data first originated from:	Stock Markets Domain	Banking and Finance Domain	c) Genomics and Astronomy	d) Social Media Domain			Genomics and Astronomy
5	Batch Processing instance is NOT an example of _____	Processing 10 GB sales data every 6	Processing flights sensor data	c) Web crawling app	a) Trending topic analysis of tweets for last 15			Trending topic analysis of tweets for last 15
6	_____ are example(s) of Real Time Big Data Processing	Complex Event Processing (CEP) platforms & Bank	Stock market data analysis	c) transactions detection	d) Stock transactions			Complex Event Processing (CEP) platforms & Bank
7	operations typically fall in the category of _____	OLTP Transactions	Big Data Batch Processing	c) Big Data Real Time Processing	d) Small Batch Processing			Big Data Real Time Processing
8	HBase used as _____	Random and Fast Read/Write operations	Faster Read only query engine in Hadoop	c) MapReduce alternative in Hadoop	d) Fast MapReduce layer in Hadoop			Random and Fast Read/Write operations in Hadoop

9	Hive used as _____	query engine	MapReduce wrapper	SQL interface	d) Hadoop
10	_____ are NOT true for Hadoop	It's a tool for Big Data analysis	supports structured and unstructure	for vertical scaling out/in scenarios	supports structured and semistructu
11	core components of Hadoop	HDFS	Map Reduce	c) HBase	d) Hive
12	Hadoop is open source _____	ALWAYS True	True only for Apache Hadoop	only for Apache and Cloudera	d) ALWAYS False
13	Hive can be used for real time queries _____	TRUE	FALSE	c) True if data set is small	some distributions
14	_____ is the default HDFS block size	32 MB	64 KB	c) 128 KB	d) 64 MB
15	default HDFS replication factor	4	1	c) 3	d) 2
16	_____ is NOT a type of metadata in NameNode	List of files	Block locations of files	c) No. of file records	access control information
17	Which of the following is/are correct	NameNode is the SPOF in Hadoop	NameNode is the SPOF in Hadoop 2.x	NameNode keeps the image of the file system	d) SPOF
18	to create replica in HDFS is _____.	Gossip protocol	Replicate protocol	c) HDFS protocol	and Forward protocol
19	NameNode tries to keep the first copy of data nearest to the client machine.	ALWAYS true	ALWAYS False	the client machine is the part of the cluster	the client machine is not the part of the
20	HDFS data blocks can be read in parallel.	TRUE	FALSE	c) data	d) read & write
21	replication factor controlled	mapred-site.xml	yarn-site.xml	c) core-site.xml	d) hdfs-site.xml
22	config files is used to define the heap size	hdfs-site.xml	core-site.xml	c) hadoop-env.sh	d) Slaves
23	_____ is not a valid Hadoop config file	mapred-site.xml	hadoop-site.xml	c) core-site.xml	d) Masters
24	NameNodes are usually high storage machines in the	True	False	c) Depends on cluster size	co-located with Job tracker

hadoop
structured and unstructured data
Map Reduce
True only for Apache Hadoop
FALSE
64 MB
3
No. of file records
keeps the image of the file system also
HDFS protocol
client machine is the part of the cluster
TRUE
hdfs-site.xml
hadoop-env.sh
hadoop-site.xml
False

25	below, select the suitable data sources for flume.	Publicly open web sites	Local data folders	c) Remote web servers	d) web services	
26	select the correct options: distcp command ALWAYS needs fully qualified	True	False	source and destination are in same cluster	source and destination are in same cluster	
27	_____ statement(s) are true about distcp command?	It invokes MapReduce in background	MapReduce if source and destination are in same cluster	c) It can't copy data from local folder to hdfs folder	can't overwrite the files through distcp command	
28	the component of Flume	Sink	Database	c) Source	d) Channel	
29	_____ is the correct sequence of MapReduce flow	Combine - Reduce - Map	Map - Combine - Reduce	c) Reduce - Combine - Map	d) Map - Reduce - Combine	
30	used to control the number of part files in a map reduce program output directory	Number of Mappers	Number of Reducers	c) Counter	d) Partitioner	
31	operations can't use Reducer as combiner	Group by Minimum	Group by Maximum	c) Group by Count	d) Group by Average	
32	_____ is/are true about combiners	Combiners can be used for mapper only job	Combiners can be used for any Map Reduce	c) Mappers can be used as a combiner class	Combiners are primarily aimed to improve	
33	Reduce side join is useful for _____	Very large datasets	Very small data sets	small and other big data sets	and other small datasets	
34	Distributed Cache can be used in _____	Mapper phase only	Reducer phase only	phase, but not on both sides simultaneously	d) In either phase	
35	Counters persist the data on hard disk.	True	False			
36	of a file for distributed cache	<=10 MB	>=250 MB	c) <=100 MB	d) <=35 MB	
37	Number of mappers is decided by the _____	specified by the programm	Available Mapper slots	c) Available heap memory	d) Input Splits	e) Input Format

Remote web servers
True
It invokes MapReduce in background
Database
Reduce - Combine - Map
Number of Reducers
Group by Average
are primarily aimed to improve Map Reduce
Very large datasets
In either phase
False
<=100 MB
Input Splits

38	joins can be performed in Reduce side join operation	Equi Join	Left Outer Join	c) Right Outer Join	d) Full Outer Join	e) All of the above	All of the above
39	limit for counters of a Map Reduce job	~5s	~15	c) ~150	d) ~50		~50
40	responsible for converting inputs to key-value Pairs of Map	FileInputFormat	InputSplit	c) RecordReader	d) Mapper		RecordReader
41	can be used to know value from a mapper/reducer	Text	IntWritable	c) NullWritable	d) String		NullWritable
44	A Map reduce job can be written in:	Java	Ruby	c) Python	Language which can read from input		Language which can read from input stream
45	Pig is a _____	Programming	Flow Language	c) Query Language	d) Database		Data Flow Language
46	Pig is good for _____	Data Factory operations	Data Warehouse operations	c) Implementing complex SQLs	Creating multiple datasets from a	e) Both (and (d)	Both (and (d)
49	correct representation to access ‘Skill’ from the Bag {‘Skills’,55, (‘Skill’, ‘Speed’), {2, (‘San’, ‘Mateo’)}}	\$3.\$1	\$3.\$0	c) \$2.\$0	d) \$2.\$1		\$3.\$1
51	for small dataset in replicated join is _____	10KB	10 MB	c) 100 MB	d) 500 MB		100 MB
52	passed to Pig scripts from _____	Parent Pig Scripts	Shell Script	Command Line	Configuration File		Command Line
53	relation can be examined through	ILLUSTRATE	DESCRIBE	c) DUMP	d) EXPLAIN		DESCRIBE
55	PigUnit tests from _____	HDFS Location	Within Program	c) HIVE	d) HBASE		HDFS Location
56	are valid Pig Control Structures	If-else	For Loop	c) Until Loop	d) Loop		Loop
57	_____ is the return data type of Filter UDF	String	Integer	c) Boolean	d) None of the above		Boolean
59	_____ of the following are not possible in Hive	Creating Tables	Creating Indexes	c) Creating Synonym	d) Writing Update Statements		Creating Synonym
60	_____ works well w	Lucene	kafka	c) MapReduce	MapReduce		MapReduce

61	set environment with multiple mongos servers, which of the following would decide the mongos	mongo shell	mongod	individual language drivers	mongos			individual language drivers
62	following SQL terminology is same as \$match in	WHERE	HAVING	Both WHERE and HAVING	GROUP BY			WHERE and HAVING
63	Which of the following is correct explanation of MongoDB processes?	mongo.exe is the shell process and mongod.exe is the actual database	mongod.exe is the shell process and mongo.exe is the actual database	mongod.exe is the MongoDB server process needed to run database	mongo.exe can be used to import database backup dump			mongo.exe is the shell process and mongod.exe is the actual database
64	mode in which the explain() command runs?	queryPlanner	executionStats	allPlansExecutor	customExecutionStats			queryPlanner
65	used as a _____, taking advantage of load balancing and data replication features over	AMS	CMS	File system	None of the mentioned			AMS
66	can be used for batch processing of data and aggregation	Hive	MapReduce	Oozie	None of the mentioned			MapReduce
67	adopted as _____ software by a number of major websites and services.	frontend	backend	proprietary	All of the mentioned			backend
68	following is a NoSQL Database	SQL	Document databases	JSON	All of the mentioned			Document databases
69	following is a wide-column store ?	Cassandra	Riak	MongoDB	Redis			Cassandra
70	following language is MongoDB written in	Javascript	C	C++	All of the mentioned			All of the mentioned
71	following sorting is not supported by	collation	collection	heap	the mentioned			collation

SYLLABUS

UNIT –V

Graph databases Neo4J- Key concept and characteristics-Modelling data for neo4j- Importing data into neo4j-Visualizations neo4j-Cypher Query Language-Data visualization- Creating Visual analytics with Tableau-Connecting your data-Creating Calculation-Using maps-Dashboard-Stories

GRAPH DATABASES NEO4J

Neo4j is the world's leading open source Graph Database which is developed using Java technology. It is highly scalable and schema free (NoSQL).

What is a Graph Database?

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. It is composed of two elements - nodes (vertices) and relationships (edges).

Graph database is a database used to model the data in the form of graph. In here, the nodes of a graph depict the entities while the relationships depict the association of these nodes.

Popular Graph Databases

Neo4j is a popular Graph Database. Other Graph Databases are Oracle NoSQL Database, OrientDB, HyperGraphDB, GraphBase, InfiniteGraph, and AllegroGraph.

Why Graph Databases?

Nowadays, most of the data exists in the form of the relationship between different objects and more often, the relationship between the data is more valuable than the data itself.

Relational databases store highly structured data which have several records storing the same type of data so they can be used to store structured data and, they do not store the relationships between the data.

Unlike other databases, graph databases store relationships and connections as first-class entities.

The data model for graph databases is simpler compared to other databases and, they can be used with OLTP systems. They provide features like transactional integrity and operational availability.

RDBMS Vs Graph Database

Following is the table which compares Relational databases and Graph databases.

Sr.No	RDBMS	Graph Database
1	Tables	Graphs
2	Rows	Nodes
3	Columns and Data	Properties and its values
4	Constraints	Relationships
5	Joins	Traversal

KEY CONCEPT AND CHARACTERISTICS

- Flexible data model – Neo4j provides a flexible simple and yet powerful data model, which can be easily changed according to the applications and industries.
- Real-time insights – Neo4j provides results based on real-time data.
- High availability – Neo4j is highly available for large enterprise real-time applications with transactional guarantees.

- Connected and semi structures data – Using Neo4j, you can easily represent connected and semi-structured data.
- Easy retrieval – Using Neo4j, you can not only represent but also easily retrieve (traverse/navigate) connected data faster when compared to other databases.
- Cypher query language – Neo4j provides a declarative query language to represent the graph visually, using an ascii-art syntax. The commands of this language are in human readable format and very easy to learn.
- No joins – Using Neo4j, it does NOT require complex joins to retrieve connected/related data as it is very easy to retrieve its adjacent node or relationship details without joins or indexes.

Features of Neo4j

Following are the notable features of Neo4j –

- Data model (flexible schema) – Neo4j follows a data model named native property graph model. Here, the graph contains nodes (entities) and these nodes are connected with each other (depicted by relationships). Nodes and relationships store data in key-value pairs known as properties.

In Neo4j, there is no need to follow a fixed schema. You can add or remove properties as per requirement. It also provides schema constraints.

- ACID properties – Neo4j supports full ACID (Atomicity, Consistency, Isolation, and Durability) rules.
- Scalability and reliability – You can scale the database by increasing the number of reads/writes, and the volume without effecting the query processing speed and data integrity. Neo4j also provides support for replication for data safety and reliability.
- Cypher Query Language – Neo4j provides a powerful declarative query language known as Cypher. It uses ASCII-art for depicting graphs. Cypher is easy to learn and can be used to create and retrieve relations between data without using the complex queries like Joins.

- Built-in web application – Neo4j provides a built-in Neo4j Browser web application. Using this, you can create and query your graph data.
- Drivers – Neo4j can work with –
 - REST API to work with programming languages such as Java, Spring, Scala etc.
 - Java Script to work with UI MVC frameworks such as Node JS.
 - It supports two kinds of Java API: Cypher API and Native Java API to develop Java applications. In addition to these, you can also work with other databases such as MongoDB, Cassandra, etc.
- Indexing – Neo4j supports Indexes by using Apache Lucence.

MODELLING DATA FOR NEO4J

Neo4j Property Graph Data Model

Neo4j Graph Database follows the Property Graph Model to store and manage its data.

Following are the key features of Property Graph Model:

- ☐ The model represents data in Nodes, Relationships and Properties
- ☐ Properties are key-value pairs
- ☐ Nodes are represented using circle and Relationships are represented using arrow keys
- ☐ Relationships have directions: Unidirectional and Bidirectional
- ☐ Each Relationship contains "Start Node" or "From Node" and "To Node" or "End Node"
- ☐ Both Nodes and Relationships contain properties
- ☐ Relationships connects nodes

In Property Graph Data Model, relationships should be directional. If we try to create relationships without direction, then it will throw an error message.

In Neo4j too, relationships should be directional. If we try to create relationships without direction, then Neo4j will throw an error message saying that "Relationships should be directional".

Neo4j Graph Database stores all of its data in Nodes and Relationships. We neither need any additional RRBMS Database nor any SQL database to store Neo4j database data. It stores its

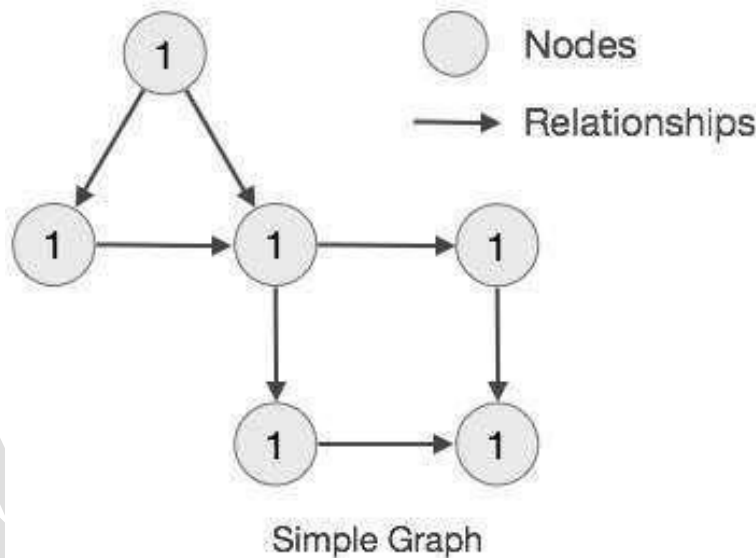
data in terms of Graphs in its native format.

Neo4j uses Native GPE (Graph Processing Engine) to work with its Native graph storage format.

The main building blocks of Graph DB Data Model are:

- ☐ Nodes
- ☐ Relationships
- ☐ Properties

Following is a simple example of a Property Graph.



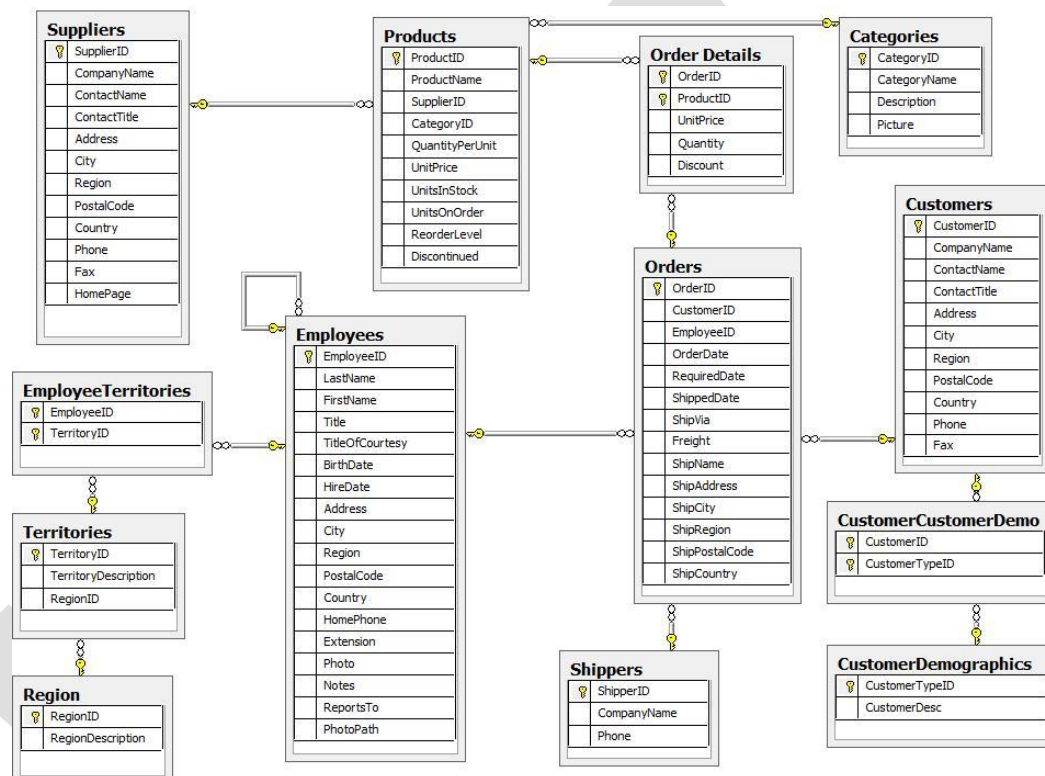
Here, we have represented Nodes using Circles. Relationships are represented using Arrows. Relationships are directional. We can represent Node's data in terms of Properties (key-value pairs). In this example, we have represented each Node's Id property within the Node's Circle.

IMPORTING DATA INTO NEO4J

NorthWind Introduction

In this guide we'll be using the NorthWind dataset, a commonly used SQL dataset. Although the NorthWind dataset is often used to demonstrate SQL and relational databases, it is graphy enough to be interesting for us.

The following is an entity relationship diagram of the NorthWind dataset:

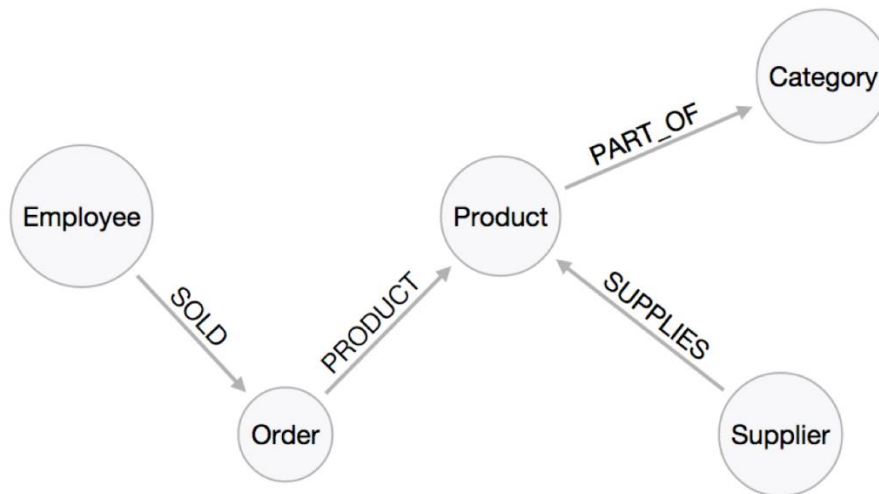


Developing a Graph Model

When deriving a graph model from a relational model, we should keep the following guidelines in mind:

- A row is a *node*
- A table name is a *label name*

In this dataset, the following graph model serves as a first iteration:



How does the Graph Model Differ from the Relational Model?

- There are no nulls.
 - In the relational version, to track the employee hierarchy we have a null entry in the 'ReportsTo' column if they don't report to anybody. In the graph version we just don't define a relationship.
 - Non existing value entries (properties) are just not present.
- It describes the relationships in more detail. For example, we know that an employee SOLD an order rather than having a foreign key relationship between the Orders and Employees tables. We could also choose to add more metadata about that relationship should we wish.
- It will often be more normalised. For example, addresses have been denormalised in several of the tables but in a future version of the graph model we might choose to make addresses nodes in their own rights.

Exporting the Data to CSV

Now that we know what we'd like our graph to look like, we need to extract the data from PostgreSQL so we can create it as a graph. The easiest way to do that is to export the appropriate tables in CSV format. The PostgreSQL 'copy' command lets us execute a SQL query and write the result to a CSV file, e.g. with `psql -d northwind < export_csv.sql`:

export_csv.sql

```
COPY (SELECT * FROM customers) TO '/tmp/customers.csv' WITH CSV header;  
COPY (SELECT * FROM suppliers) TO '/tmp/suppliers.csv' WITH CSV header;  
COPY (SELECT * FROM products) TO '/tmp/products.csv' WITH CSV header;  
COPY (SELECT * FROM employees) TO '/tmp/employees.csv' WITH CSV header;  
COPY (SELECT * FROM categories) TO '/tmp/categories.csv' WITH CSV header;
```

```
COPY (SELECT * FROM orders  
LEFT OUTER JOIN order_details ON order_details.OrderID = orders.OrderID) TO  
'/tmp/orders.csv' WITH CSV header;
```

Importing the Data using Cypher

After we've exported our data from PostgreSQL, we'll use Cypher's LOAD CSV command to transform the contents of the CSV file into a graph structure.

First, create the nodes:

import_csv.cypher

```
// Create customers  
USING PERIODIC COMMIT  
LOAD CSV WITH HEADERS FROM "file:customers.csv" AS row  
CREATE (:Customer {companyName: row.CompanyName, customerID: row.CustomerID, fax:  
row.Fax, phone: row.Phone});
```

```
// Create products  
USING PERIODIC COMMIT  
LOAD CSV WITH HEADERS FROM "file:products.csv" AS row  
CREATE (:Product {productName: row.ProductName, productID: row.ProductID, unitPrice:  
toFloat(row.UnitPrice)});
```

```
// Create suppliers  
USING PERIODIC COMMIT  
LOAD CSV WITH HEADERS FROM "file:suppliers.csv" AS row  
CREATE (:Supplier {companyName: row.CompanyName, supplierID: row.SupplierID});
```

```
// Create employees
```

USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:employees.csv" AS row

```
CREATE (:Employee {employeeID:row.EmployeeID, firstName: row.FirstName, lastName:
row.LastName, title: row.Title});
```

// Create categories

USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:categories.csv" AS row

```
CREATE (:Category {categoryID: row.CategoryID, categoryName: row.CategoryName,
description: row.Description});
```

USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:orders.csv" AS row

```
MERGE (order:Order {orderID: row.OrderID}) ON CREATE SET order.shipName =
row.ShipName;
```

Next, we'll create indexes on the just-created nodes to ensure their quick lookup when creating relationships in the next step.

```
CREATE INDEX ON :Product(productID);
```

```
CREATE INDEX ON :Product(productName);
```

```
CREATE INDEX ON :Category(categoryID);
```

```
CREATE INDEX ON :Employee(employeeID);
```

```
CREATE INDEX ON :Supplier(supplierID);
```

```
CREATE INDEX ON :Customer(customerID);
```

```
CREATE INDEX ON :Customer(customerName);
```

```
CREATE CONSTRAINT ON (o:Order) ASSERT o.orderID IS UNIQUE;
```

As the indexes are created after the nodes are inserted, their population happens asynchronously, so we use `schema await` (a shell command) to block until they are populated.

`schema await`

Initial nodes and indices in place, we can now create relationships of orders to products and employees.

USING PERIODIC COMMIT

LOAD CSV WITH HEADERS FROM "file:orders.csv" AS row

```
MATCH (order:Order {orderID: row.OrderID})
MATCH (product:Product {productID: row.ProductID})
MERGE (order)-[pu:PRODUCT]->(product)
ON CREATE SET pu.unitPrice = toFloat(row.UnitPrice), pu.quantity = toFloat(row.Quantity);
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:orders.csv" AS row
MATCH (order:Order {orderID: row.OrderID})
MATCH (employee:Employee {employeeID: row.EmployeeID})
MERGE (employee)-[:SOLD]->(order);
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:orders.csv" AS row
MATCH (order:Order {orderID: row.OrderID})
MATCH (customer:Customer {customerID: row.CustomerID})
MERGE (customer)-[:PURCHASED]->(order);
```

Next, create relationships between products, suppliers, and categories:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:products.csv" AS row
MATCH (product:Product {productID: row.ProductID})
MATCH (supplier:Supplier {supplierID: row.SupplierID})
MERGE (supplier)-[:SUPPLIES]->(product);
```

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:products.csv" AS row
MATCH (product:Product {productID: row.ProductID})
MATCH (category:Category {categoryID: row.CategoryID})
MERGE (product)-[:PART_OF]->(category);
```

Finally we'll create the 'REPORTS_TO' relationship between employees to represent the reporting structure:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:employees.csv" AS row
MATCH (employee:Employee {employeeID: row.EmployeeID})
MATCH (manager:Employee {employeeID: row.ReportsTo})
```


MERGE (employee)-[:REPORTS_TO]->(manager);

You can also run the whole script at once using `bin/neo4j-shell -path northwind.db -file import_csv.cypher`.

Querying the Graph

One question we might be interested in is:

Which Employee had the Highest Cross-Selling Count of 'Chocolade' and Which Product?

```
MATCH (choc:Product {productName:'Chocolade'})<-[:PRODUCT]-(Order)<-[:SOLD]-(employee),
      (employee)-[:SOLD]->(o2)-[:PRODUCT]->(other:Product)
RETURN employee.employeeID, other.productName, count(distinct o2) as count
ORDER BY count DESC
LIMIT 5;
```

Looks like employee No. 1 was very busy!

employee.employeeId	other.productName	count
1	Pavlova	56
1	Camembert Pierrot	56
1	Ikura	55
1	Chang	47
1	Pâté chinois	45

We might also like to answer the following question:

How are Employees Organized? Who Reports to Whom?

```
MATCH path = (e:Employee)<-[:REPORTS_TO]-(sub)
RETURN e.employeeID AS manager, sub.employeeID AS employee;
manager employee
```

manager	employee
2	1
2	3
2	4
2	5
2	8
5	6
5	7
5	9

Notice that employee No. 5 has people reporting to them but also reports to employee No. 2.

Let's investigate that a bit more:

Which Employees Report to Each Other Indirectly?

```
MATCH path = (e:Employee)<-[:REPORTS_TO*]-(sub)
WITH e, sub, [person in NODES(path) | person.employeeID][1..-1] AS path
RETURN e.employeeID AS manager, sub.employeeID AS employee, CASE WHEN
LENGTH(path) = 0 THEN "Direct Report" ELSE path END AS via
ORDER BY LENGTH(path);
```

e.EmployeeID	sub.EmployeeID	via
2	1	Direct Report
2	3	Direct Report
2	4	Direct Report
2	5	Direct Report
2	8	Direct Report
5	6	Direct Report
5	7	Direct Report
5	9	Direct Report
2	6	[5]
2	7	[5]
2	9	[5]

How Many Orders were Made by Each Part of the Hierarchy?

MATCH (e:Employee)

OPTIONAL MATCH (e)-[:REPORTS_TO*0..]-(:sub)-[:SOLD]->(order)

RETURN e.employeeID, [x IN COLLECT(DISTINCT sub.employeeID) WHERE x <> e.employeeID] AS reports, COUNT(distinct order) AS totalOrders
ORDER BY totalOrders DESC;

e.EmployeeID	reports	totalOrders
2	[1,3,4,5,6,7,9,8]	2155
5	[6,7,9]	568
4	[]	420
1	[]	345
3	[]	321
8	[]	260
7	[]	176
6	[]	168
9	[]	107

Updating the Graph

Now if we wanted to update our graph data, we have to first find the relevant information and then update or extend the graph structures.

Janet is now reporting to Steven

We need to find Steven first, and Janet and her **REPORTS_TO** relationship. Then we remove the existing relationship and create a new one to Steven.

MATCH (mgr:Employee {EmployeeID:5})

MATCH (emp:Employee {EmployeeID:3})-[rel:REPORTS_TO]->()

DELETE rel

CREATE (emp)-[:REPORTS_TO]->(mgr)

RETURN *;

This single relationship change is all you need to update a part of the organizational hierarchy. All subsequent queries will immediately use the new structure.

GRAPH VISUALIZATIONS:

Graph visualisations are a powerful tool to convey the content of a graph. They can highlight patterns, and show clusters and connections. There are many excellent options for graph visualization, such as [D3.js](#), [three.js](#), [sigma.js](#) and [Alchemy.js](#). A few of the more popular graph-visualization methods are discussed below.

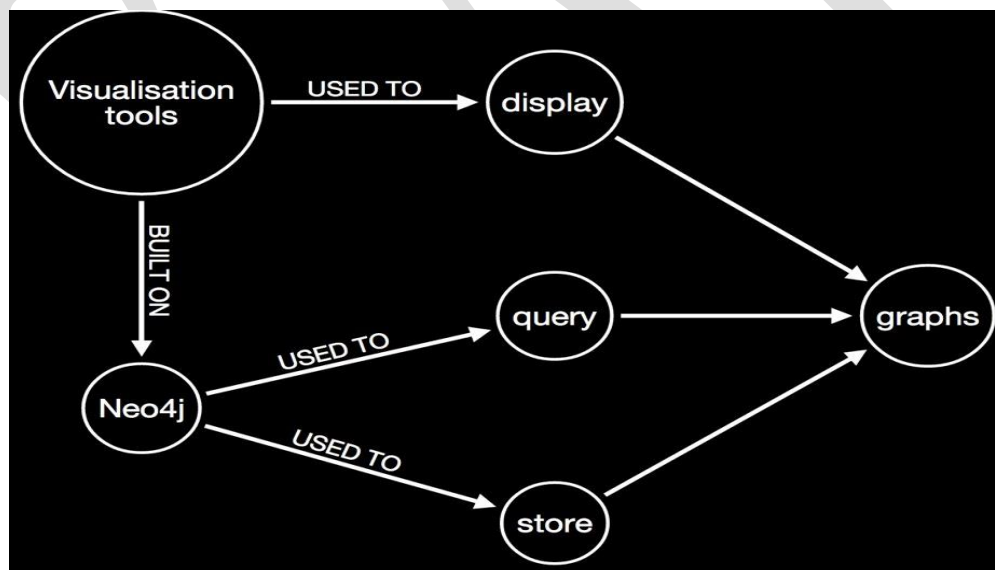
In all cases, the visualizer request JSON data for all or part of the graph data then dynamically creates an in-memory JavaScript visualization on the client side.

Screencast: The Neo4j Browser

The default [Neo4j Server](#) has a powerful, customizable data visualization tool based on the built-in D3.js library.

In the screencast, we demonstrate how to style nodes and relationships in the Neo4j's Browser visualization, and how to set colors, sizes, and titles. We then discuss the Graph-Style-Sheet (GRASS) and how you can download, update, and reset the styling information.

Presentation: SVG-Based Graph Interaction



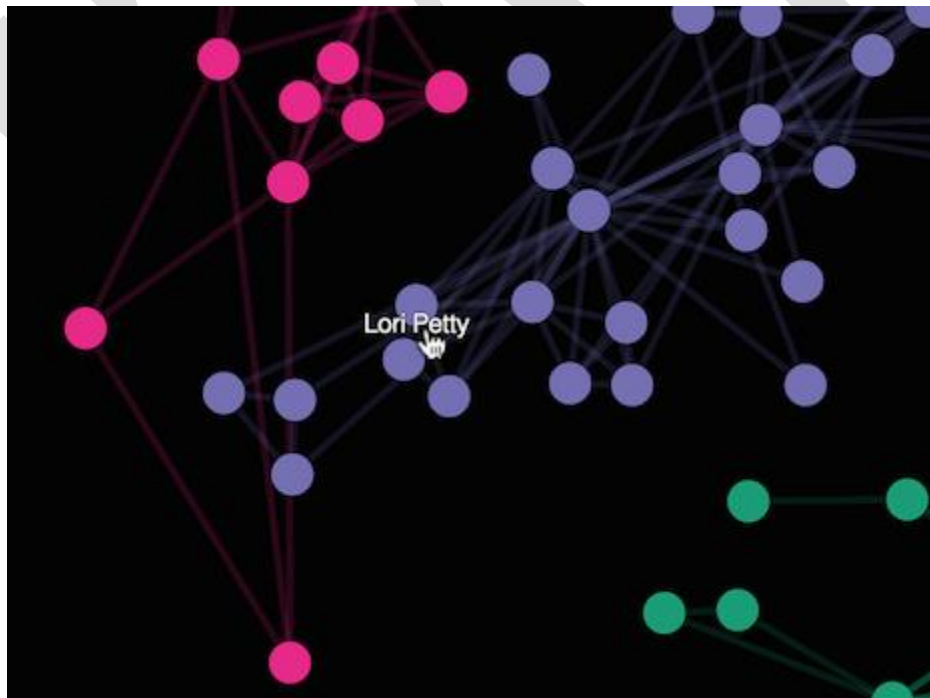
In this talk, Alistair Jones builds a very powerful graph web editing tool based on SVG, suited for easy integration into modern HTML frameworks.

Library: Alchemy.js Open Source Graph Visualization

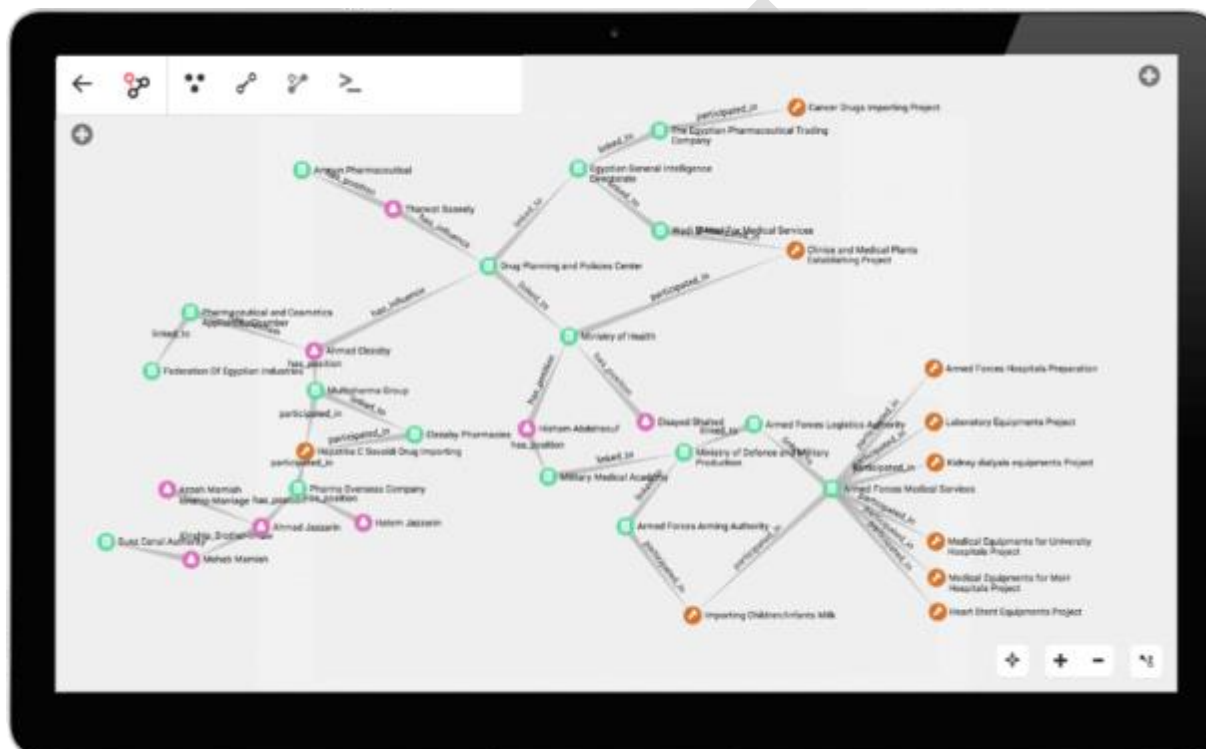
As you've just seen, preparing, converting and rendering graph data with plain D3 force layout involves a bit of work. The open-source Alchemy.js library replaces this effort with a few lines of configuration. It uses the GraphJSON format for interaction which is easier to handle than the raw D3 JSON format. Alchemy.js is mostly configuration based but can also be customized with javascript functions. It supports custom styling, highlighting, hover effects, clustering and interactive exploration.

Here is a simple example of using Alchemy.js to render a simple graph:

```
<script src="http://cdn.graphalchemist.com/alchemy.min.js"></script>
<script type="text/javascript">
  alchemy.begin({
    dataSource: "actors.json",
    nodeCaption: 'name',
    nodeMouseOver: 'name',
    cluster: true,
    clusterColours: ["#1B9E77", "#D95F02", "#7570B3", "#E7298A", "#66A61E", "#E6AB02"]})
</script>
```



Product: Linkurious Graph Visualization



Product: Linkurious Enterprise

Linkurious is an on-premise graph visualization and analysis software that allows you to detect suspicious patterns, understand connections and discover new insights in your data.

Linkurious Enterprise leverages Neo4j's graph database technology to offer you an easy solution to search, visualize, detect patterns and edit graph data. You can easily search for properties, nodes and relationships thanks to a built-in search engine or monitor specific Cypher query alerts. Linkurious also optimizes multi-users collaboration and make sharing and reporting available to teams. Linkurious is used in many different fields from fraud detection to enterprise architecture management or IT management. The solution provides a security framework compatible with the security requirements of Fortune 500 companies and government agencies.

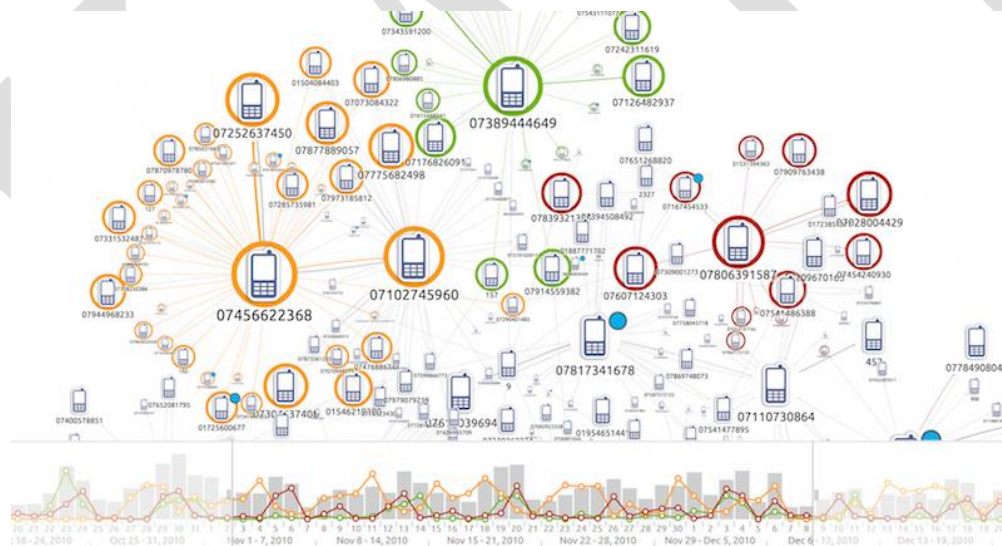
Product: Linkurious SDK

The Linkurious SDK is a powerful toolkit to build graph visualization and analysis applications or add graph features to existing apps. The Linkurious Server REST API provides access to high-level services on top of Neo4j including data access (full text search or graph query), authentication and access rights management, alert management or visualization management (create, share, collaborate). The Ogma JavaScript graph visualizations library provides the ability to build interactive scalable graph visualization user interfaces. Display information with various graph layouts (Grid, Concentric, ForceLink, Continuous or Hierarchical) for a fast understanding of the data or customize the style (colors, icons, size, shape) to obtain easy-to-read graph visualizations.

Product: Tom Sawyer Perspectives

Tom Sawyer Perspectives 6.0 offers Cypher support for Neo4j graph databases and connects them to the broad visualization capabilities of the tool. Tom Sawyer Perspectives is integrated with the latest Neo4j version for easy visualization of complex results.

Product: Keylines Neo4j Graph Visualization



- How to Visualize Time-Based Graphs with Neo4j

KeyLines is an out-of-the-box JavaScript solution for visualizing networks. It works in all major browsers and all platforms, including the iPad.

KeyLines integrates into existing web applications easily and with very little effort. How and where you get the data is up to you. KeyLines does the job of rendering it and responding to user interactions like clicking, touching, moving nodes, and more. You bind to these events to customize what happens, and, most importantly, your data stays under your control at all times: KeyLines is self-contained and needs no external connections.

HowTo: Graph Visualization Step By Step

If you want to get started directly, here are the few steps you have to take to convert Neo4j's graph query results into the format typically needed by the visualization libraries. This approach is also used in the example project which we implemented for each of the languages and drivers listed. You can find the appropriate transformation code for your language and stack there.

Target JSON Datastructure

They visualization toolkits usually expect a JSON structure of node objects (their id, label(s) and properties) and a list of relationships (their properties, start- and end-node-id (or node-array-index for d3) and optionally relationship-id) like this. Here we see 3 nodes (2 people, one database) and 3 relationships in that datastructure.

```
{ "nodes": [{ name: "Peter", label: "Person", id: 1 }, { name: "Michael", label: "Person", id: 2 },  
  { name: "Neo4j", label: "Database", id: 3 } ],  
  "links": [{ source: 0, target: 1, type: "KNOWS", since: 2010 }, { source: 0, target: 2,  
    type: "FOUNDED" },  
    { source: 1, target: 2, type: "WORKS_ON" } ] }
```

Graph Rendering Javascript Code

To render this list with D3, you just need the d3.js library dependency and a few lines of javascript.

Unresolved directive in <stdin> - include::.../..language-guides/assets/index.html[tags=d3]

Neo4j Query Result Format

To generate the expected format from Neo4j you convert the results returned from the HTTP Cypher API endpoint. That endpoint already has an option (`{"resultDataContents":["graph"]}`) to return graph only results.

For instance executing this request:

```
:POST /db/data/transaction/commit
{"statements":[{"statement":"MATCH path = (n)-[r]->(m) RETURN path",
  "resultDataContents":["graph"]}]}
```

results in:

```
var res =
{ "results": [
  {
    "columns": ["path"],
    "data" : [{
      "graph": {
        "nodes": [
          {"id": "1", "labels": ["Person"], "properties": {"name": "Peter"}},
          {"id": "2", "labels": ["Person"], "properties": {"name": "Michael"}}
        ],
        "relationships": [
          {"id": "0", "type": "KNOWS", "startNode": "1", "endNode": "2", "properties": {}}
        ]
      } // , {"graph": ...}, ...
    ]}
  ], "errors": []
}
```

Converting Neo4j Query Results to D3 JSON

Which can be converted to our visualization format by collecting the nodes and relationships of the different rows into two lists. Just make sure that each node is unique in the list and that the relationships refer to the array-index (not the id) of the node.

```
function idIndex(a,id) {
  for (var i=0;i<a.length;i++) {
    if (a[i].id == id) return i;}
  return null;
}
var nodes=[], links=[];
res.results[0].data.forEach(function (row) {
```

```
row.graph.nodes.forEach(function (n) {  
  if (idIndex(nodes,n.id) == null)  
    nodes.push({ id:n.id,label:n.labels[0],title:n.properties.name });  
});  
links = links.concat( row.graph.relationships.map(function(r) {  
  return { start:idIndex(nodes,r.startNode),end:idIndex(nodes,r.endNode),type:r.type };  
}));  
});  
viz = {nodes:nodes, links:links};
```

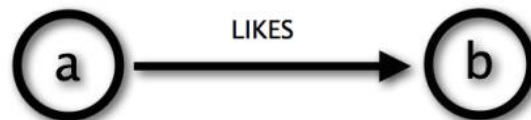
CYPHER QUERY LANGUAGE

About Cypher

Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax.

It allows us to state what we want to select, insert, update or delete from our graph data without requiring us to describe exactly how to do it.

Cypher using relationship 'likes'



Cypher

(a) -[:LIKES]-> (b)

Nodes

Cypher uses ASCII-Art to represent patterns. We surround nodes with parentheses which look like circles, e.g. (node). If we later want to refer to the node, we'll give it an variable like (p) for person or (t) for thing. In real-world queries, we'll probably use longer, more expressive variable

names like (person) or (thing). If the node is not relevant to your question, you can also use empty parentheses ().

Usually, the relevant labels of the node are provided to distinguish between entities and optimize execution, like (p:Person).

We might use a pattern like (person:Person)-->(thing:Thing) so we can refer to them later, for example, to access properties like person.name and thing.quality.

The more general structure is:

MATCH (node:Label) RETURN node.property

MATCH (node1:Label1)-->(node2:Label2)

WHERE node1.propertyA = {value}

RETURN node2.propertyA, node2.propertyB

- [SQL to Cypher](#)
- [Free Online Training](#)
- [Cypher Reference Card](#)
- [Manual: Cypher Reference](#)

Please note that *node-labels*, *relationship-types* and *property-names* are case-sensitive in Cypher. All the other clauses, keywords and functions are not, but should be cased consistently according to the style used here.

Relationships

To fully utilize the power of our graph database we want to express more complex patterns between our nodes. Relationships are basically an arrow --> between two nodes. Additional information can be placed in square brackets inside of the arrow.

This can be

- relationship-types like -[:KNOWS|:LIKE]->
- a variable name -[rel:KNOWS]-> before the colon
- additional properties -[{{since:2010}}]->
- structural information for paths of variable length -[:KNOWS*..4]->

To access information about a relationship, we can assign it a variable, for later reference. It is placed in front of the colon `-[rel:KNOWS]->` or stands alone `-[rel]->`.

If you forget the colon in front of a relationship-type, like this `-[KNOWS]->` it does represent a variable and the relationship has no relationship-type declared.

General Syntax:

```
MATCH (n1:Label1)-[rel:TYPE]->(n2:Label2)
WHERE rel.property > { value }
RETURN rel.property, type(rel)
```

Patterns

Nodes and relationship expressions are the building blocks for more complex patterns. Patterns can be written continuously or separated with commas. You can refer to variables declared earlier or introduce new ones.

- friend-of-a-friend `(user)-[:KNOWS]-(friend)-[:KNOWS]-(foaf)`
- shortest path: `path = shortestPath((user)-[:KNOWS*..5]-(other))`
- collaborative filtering `(user)-[:PURCHASED]->(product)<-[:PURCHASED]-()-[:PURCHASED]->(otherProduct)`
- tree navigation `(root)<-[:PARENT*]-(leaf:Category)-[:ITEM]->(data:Product)`

Patterns can be used to `MATCH` and `CREATE` data, but also (evaluating to a list of paths) in expressions, predicates and results.

Let's try out what we've learned. See [an interactive live example](#) of the exercise below.

First Steps with Cypher

Create a Record for Yourself

```
CREATE (you:Person { name:"You" })
RETURN you
```

`CREATE` creates nodes with labels and properties.

You like *Neo4j*, right?

Let's find ourselves and add a new relationship to a new node.

```
MATCH (you:Person {name:"You"})
CREATE (you)-[like:LIKE]->(neo:Database {name:"Neo4j" })
RETURN you,like,neo
```

CREATE can create single nodes, or more complex structures.

Create Your Friends

```
MATCH (you:Person {name:"You"})
FOREACH (name in ["Johan","Rajesh","Anna","Julia","Andrew"] |
  CREATE (you)-[:FRIEND]->(:Person {name:name}))
```

FOREACH allows you to execute update operations for each element of a list.

Find Your Friends

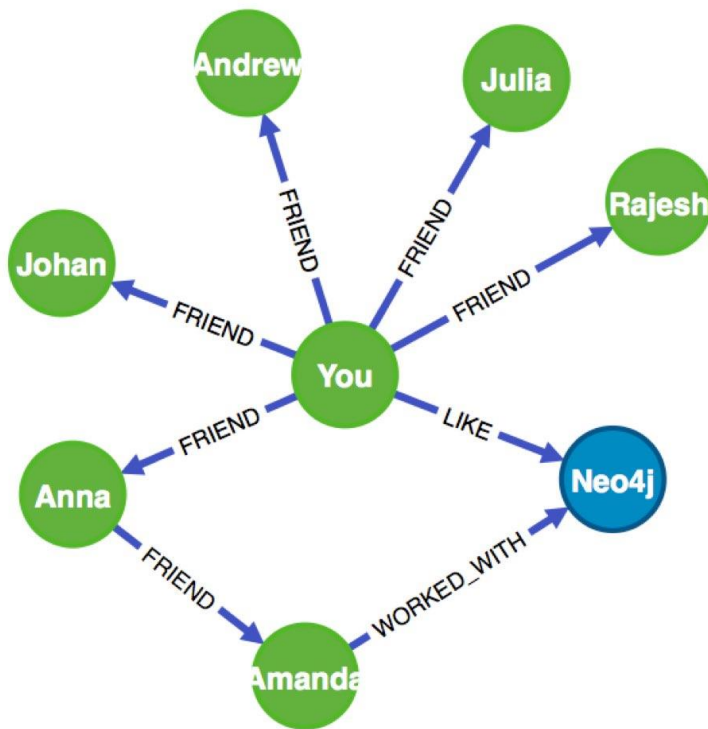
```
MATCH (you {name:"You"})-[:FRIEND]->(yourFriends)
RETURN you, yourFriends
```

Note that we get ourselves repeated for each path found in the graph.

Create Second Degree Friends and Expertise

```
MATCH (neo:Database {name:"Neo4j"})
MATCH (anna:Person {name:"Anna"})
CREATE (anna)-[:FRIEND]->(:Person:Expert {name:"Amanda"})-[:WORKED_WITH]->(neo)
```

CREATE can also add more complex patterns.



Find Someone in your Network Who Can Help You Learn Neo4j

```
MATCH (you {name:"You"})
```

```
MATCH (expert)-[:WORKED_WITH]->(db:Database {name:"Neo4j"})
```

```
MATCH path = shortestPath( (you)-[:FRIEND*..5]-(expert) )
```

```
RETURN db,expert,path
```

DATA VISUALIZATION:

Data Visualization is an art of presenting the data in a manner that even a non-analyst can understand it. A perfect blend of aesthetic elements like colors, dimensions, labels can create visual masterpieces, hence revealing surprising business insights which in turn helps businesses to make informed decisions.

Data Visualization is an inevitable aspect of business analytics. As more and more sources of data are getting discovered, business managers at all levels embrace data visualization softwares, that allow them to analyze trends visually and take quick decisions. Currently, the most popular tools for visualizations / data discovery are Qlikview and Tableau. We have already published a step by step learning path for Qlikview.

Tableau is one of the fastest evolving Business Intelligence (BI) and data visualization tool. It is very fast to deploy, easy to learn and very intuitive to use for a customer. Here is a learning path to all those people who are new to Tableau. This path will help you to learn Tableau in a structured approach. Beginners are recommended to follow this path religiously. If you already have some background, or don't need all the components, feel free to follow your own paths and let us know how did it turn out to be!

CONNECTING WITH DATA

Tableau can connect with various data sources such as text, excel file, databases to big data queries also. In this section, we will look at the basics and advance feature of data connectivity with different sources. Here we will also look at Join types, Data Blending, connection with cubes, custom sql and Google Analytics.

CREATING CALCULATION

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Tableau has a number of operators used to create calculated fields and formulas.

Following are the details of the operators that are available and the order (precedence) of operations.

Types of Operator

- General Operators
- Arithmetic Operators
- Relational Operators
- Logical Operators

General Operators

Following table shows the general operators supported by Tableau. These operators act on numeric, character, and date data types.

Operator	Description	Example
+(addition)	Adds two numbers. Concatenates two strings. Adds days to dates.	$7 + 3$ Profit + Sales 'abc' + 'def' = 'abcdef' #April 15, 2004# + 15 = #April 30, 2004#
– (subtraction)	Subtracts two numbers. Subtracts days from dates.	$-(7+3) = -10$ #April 16, 2004# - 15 = #April 1, 2004#

Arithmetic Operators

Following table shows the arithmetic operators supported by Tableau. These operators act only on numeric data types.

Operator	Description	Example
*(Multiplication)	Numeric multiplication	$23 * 2 = 46$
/(Division)	Numeric division	$45 / 2 = 22.5$

%(modulo)	Reminder of numeric division	13 % 2 = 1
^(power)	Raised to the power	2^3 = 8

Comparison Operators

Following table lists the comparison operators supported by Tableau. These operators are used in expressions. Each operator compares two numbers, dates, or strings and returns a Boolean (TRUE or FALSE). Booleans themselves, however, cannot be compared using these operators.

Operator	Description	Example
= or = (Equal to)	Compares two numbers or two strings or two dates to be equal. Returns the Boolean value TRUE if they are, else returns false.	'Hello' = 'Hello' 5 = 15/3
!= or <> (Not equal to)	Compares two numbers or two strings or two dates to be unequal. Returns the Boolean value TRUE if they are, else returns false.	'Good' <> 'Bad' 18 != 37/2
> (Greater than)	Compares two numbers or two strings or two dates where the first argument is greater than second. Returns the boolean value TRUE if it is the case, else returns false.	[Profit] > 20000 [Category] > 'Q' [Ship date] > #April 1, 2004#
< (Less than)	Compares two numbers or two strings or two dates where the first argument is smaller than second. Returns the boolean value TRUE if it is the case, else returns false.	[Profit] < 20000 [Category] < 'Q' [Ship date] < #April 1, 2004#

Logical Operators

Following table shows the logical operators supported by Tableau. These operators are used in expressions whose result is a Boolean giving the output as TRUE or FALSE.

Operator	Description	Example
AND	If the expressions or Boolean values present on both sides of AND operator is evaluated to be TRUE, then the result is TRUE. Else the result is FALSE.	[Ship Date] > #April 1, 2012# AND [Profit] > 10000
OR	If any one or both of the expressions or Boolean values present on both sides of AND operator is evaluated to be TRUE, then the result is TRUE. Else the result is FALSE.	[Ship Date] > #April 1, 2012# OR [Profit] > 10000
NOT	This operator negates the Boolean value of the expression present after it.	NOT [Ship Date] > #April 1, 2012#

Operator Precedence

The following table describes the order in which operators are evaluated. The top row has the highest precedence. Operators on the same row have the same precedence. If two operators have the same precedence, they are evaluated from left to right in the formula. Also parentheses can be used. The inner parentheses are evaluated before the outer parentheses.

Precedence	Operator
1	–(negate)
2	^(power)
3	*, /, %
4	+, –

5	==, >, <, >=, <=, !=
6	NOT
7	AND
8	OR

FUNCTIONS:

Any data analysis involves a lot of calculations. In Tableau, the calculation editor is used to apply calculations to the fields being analyzed. Tableau has a number of inbuilt functions which help in creating expressions for complex calculations.

Following are the description of different categories of functions.

- Number Functions
- String Functions
- Date Functions
- Logical Functions
- Aggregate Functions

Number Functions

These are the functions used for numeric calculations. They only take numbers as inputs. Following are some examples of important number functions.

Function	Description	Example
CEILING (number)	Rounds a number to the nearest integer of equal or greater value.	CEILING(2.145) = 3

POWER (number, power)	Raises the number to the specified power.	POWER(5,3) = 125
ROUND (number, [decimals])	Rounds the numbers to a specified number of digits.	ROUND(3.14152,2) = 3.14

String Functions

String Functions are used for string manipulation. Following are some important string functions with examples

Function	Description	Example
LEN (string)	Returns the length of the string.	LEN("Tableau") = 7
LTRIM (string)	Returns the string with any leading spaces removed.	LTRIM(" Tableau ") = "Tableau"
REPLACE (string, substring, replacement)	Searches the string for substring and replaces it with a replacement. If the substring is not found, the string is not changed.	REPLACE("GreenBlueGreen", "Blue", "Red") = "GreenRedGreen"
UPPER (string)	Returns string, with all characters uppercase.	UPPER("Tableau") = "TABLEAU"

Date Functions

Tableau has a variety of date functions to carry out calculations involving dates. All the date functions use the date_part which is a string indicating the part of the date such as - month, day, or year. Following table lists some examples of important date functions.

Function	Description	Example
DATEADD (date_part, increment, date)	Returns an increment added to the date. The type of increment is specified in date_part.	DATEADD ('month', 3, #2004-04-15#) = 2004-07-15 12:00:00 AM
DATENAME (date_part, date, [start_of_week])	Returns date_part of date as a string. The start_of_week parameter is optional.	DATENAME('month', #2004-04-15#) = "April"
DAY (date)	Returns the day of the given date as an integer.	DAY(#2004-04-12#) = 12
NOW()	Returns the current date and time.	NOW() = 2004-04-15 1:08:21 PM

Logical Functions

These functions evaluate some single value or the result of an expression and produce a boolean output.

Function	Description	Example
IFNULL (expression1, expression2)	The IFNULL function returns the first expression if the result is not null, and returns the second expression if it is null.	IFNULL([Sales], 0) = [Sales]
ISDATE (string)	The ISDATE function returns TRUE if the string argument can be converted to a date, and FALSE if it cannot.	ISDATE("11/05/98") = TRUE ISDATE("14/05/98") =

		FALSE
MIN(expression)	The MIN function returns the minimum of an expression across all records or the minimum of two expressions for each record.	

Aggregate Functions

Function	Description	Example
AVG(expression)	Returns the average of all the values in the expression. AVG can be used with numeric fields only. Null values are ignored.	
COUNT(expression)	Returns the number of items in a group. Null values are not counted.	
MEDIAN(expression)	Returns the median of an expression across all records. Median can only be used with numeric fields. Null values are ignored.	
STDEV(expression)	Returns the statistical standard deviation of all values in the given expression based on a sample of the population.	

NUMERIC CALCULATIONS

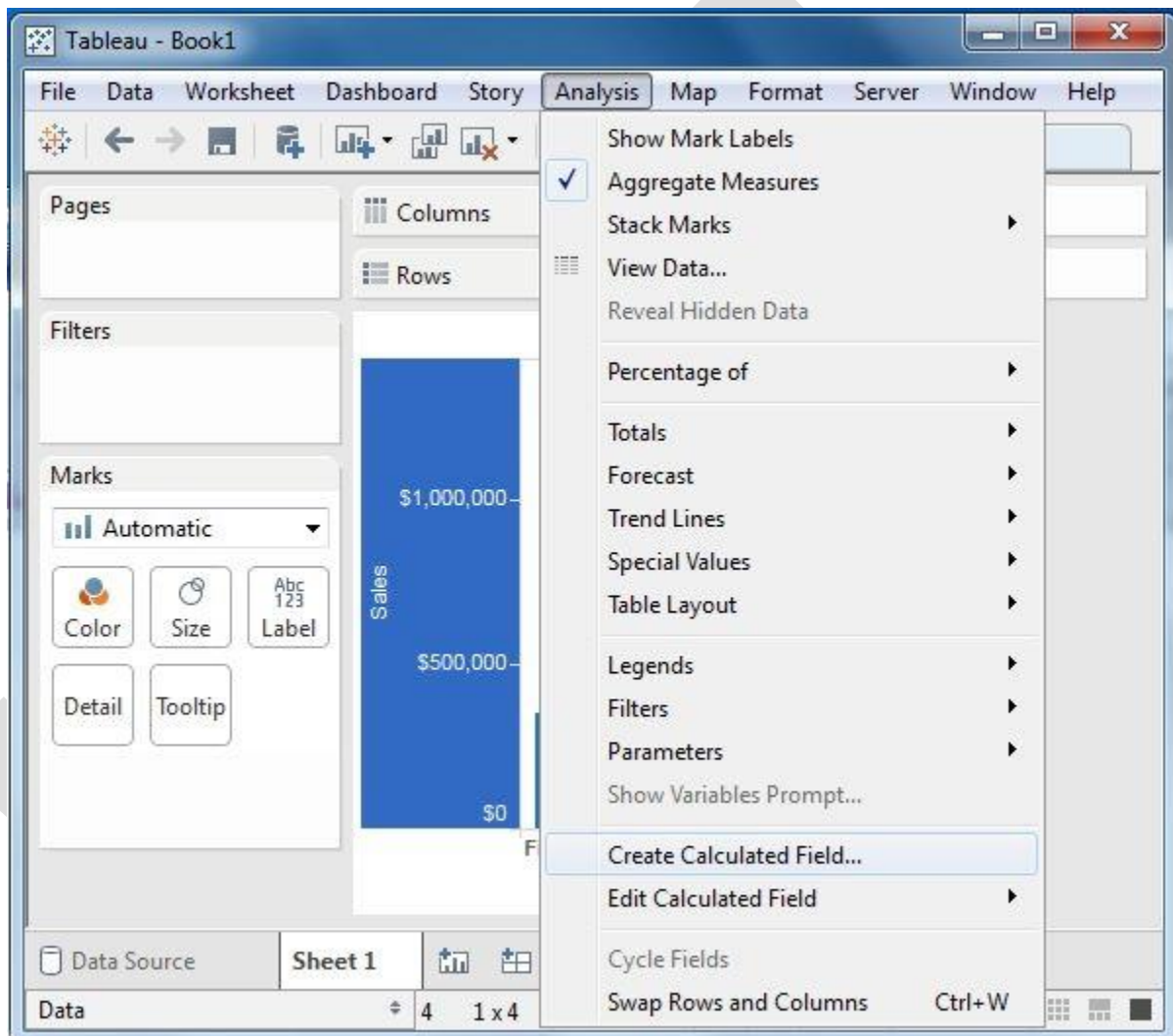
Numeric calculations in Tableau are done using a wide range of inbuilt functions available in the formula editor.

In this chapter, we will see how to apply calculations to the fields. The calculations can be as simple as subtracting the values of two fields or applying an aggregate function to a single field.

Following are the steps to create a calculation field and use numeric functions in it.

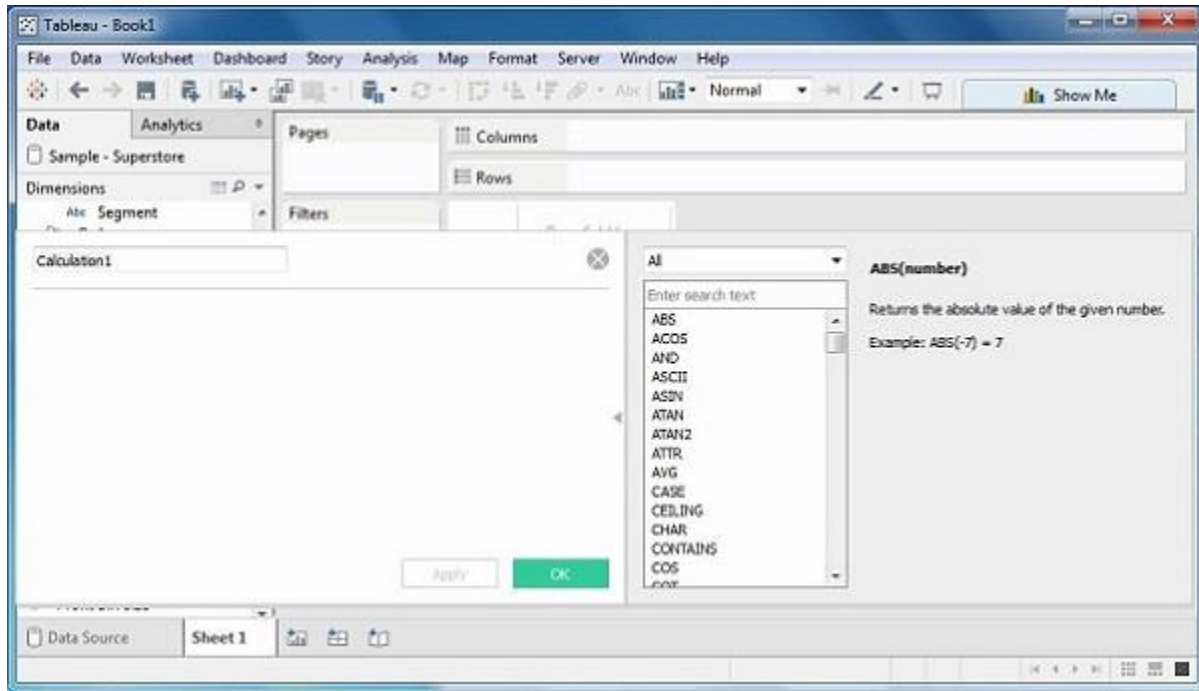
Create Calculated Field

While connected to Sample-superstore, go to the Analysis menu and click 'Create Calculated Field', as shown in the following screenshot.



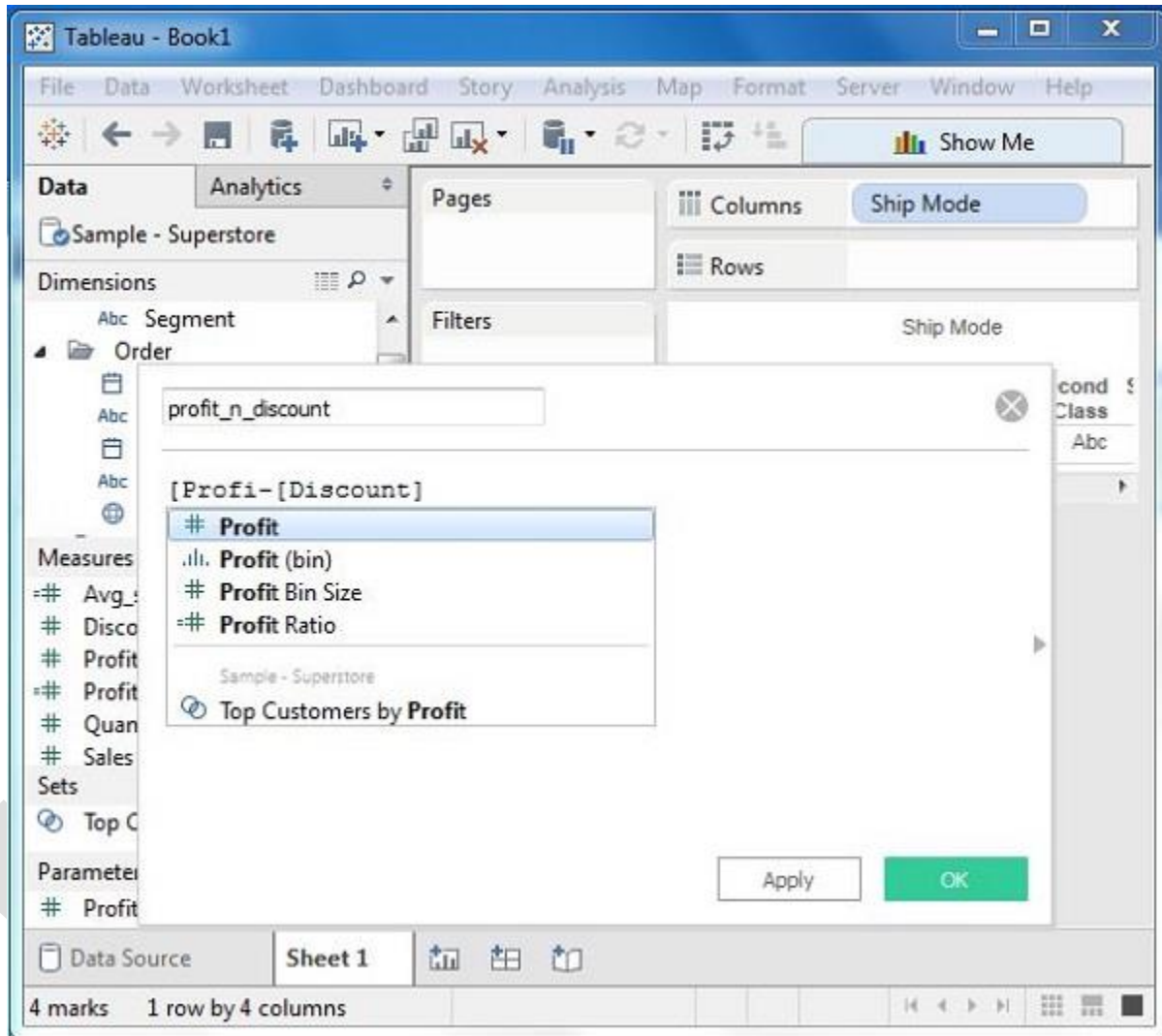
Calculation Editor

The above step opens a calculation editor which lists all the functions that are available in Tableau. You can change the dropdown value and see only the functions related to numbers.



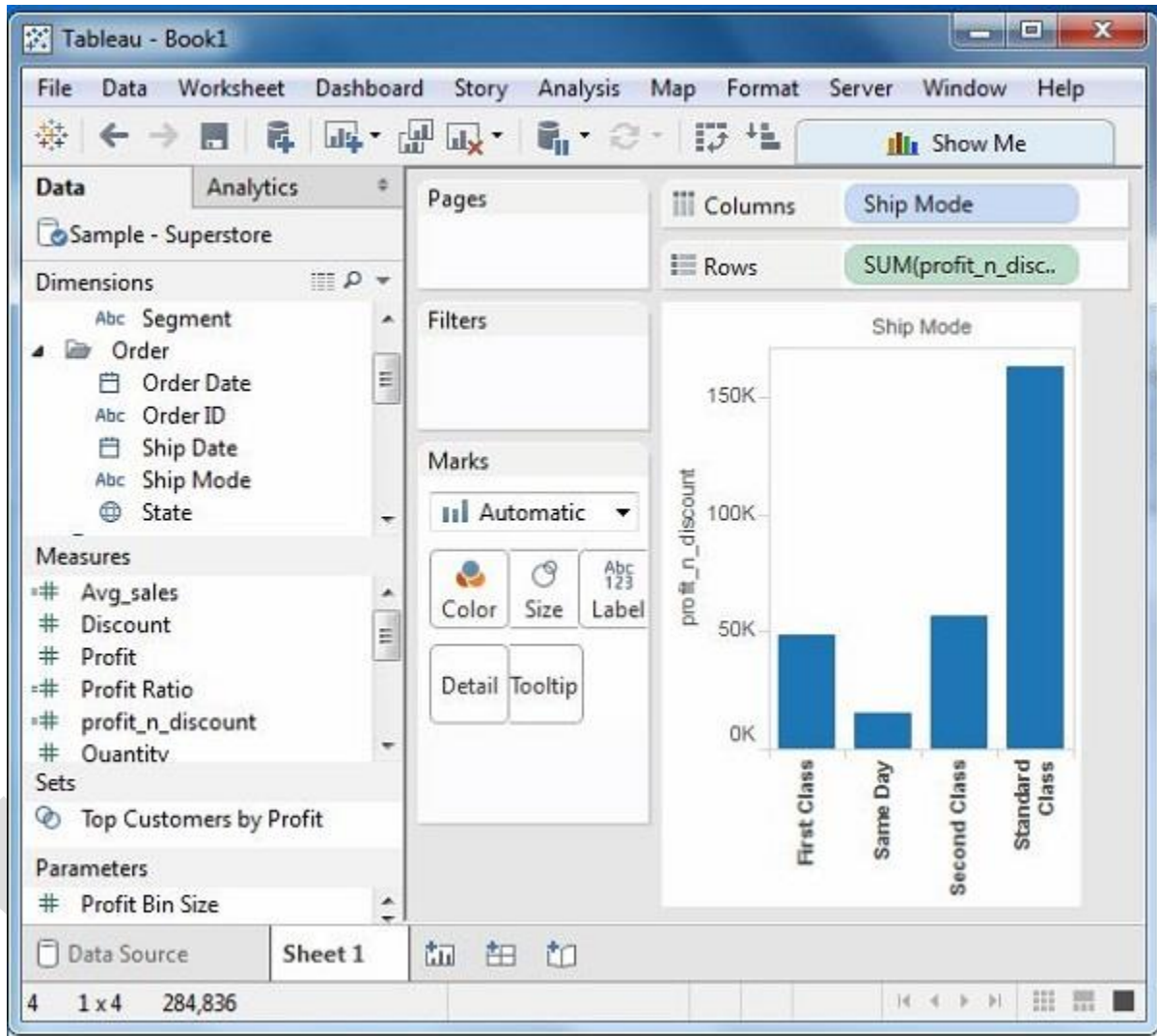
Create a Formula

To study the difference between profit and discount for different shipping mode of the products, create a formula subtracting the discount from the profit as shown in the following screenshot. Also, name this field as profit_n_discount.



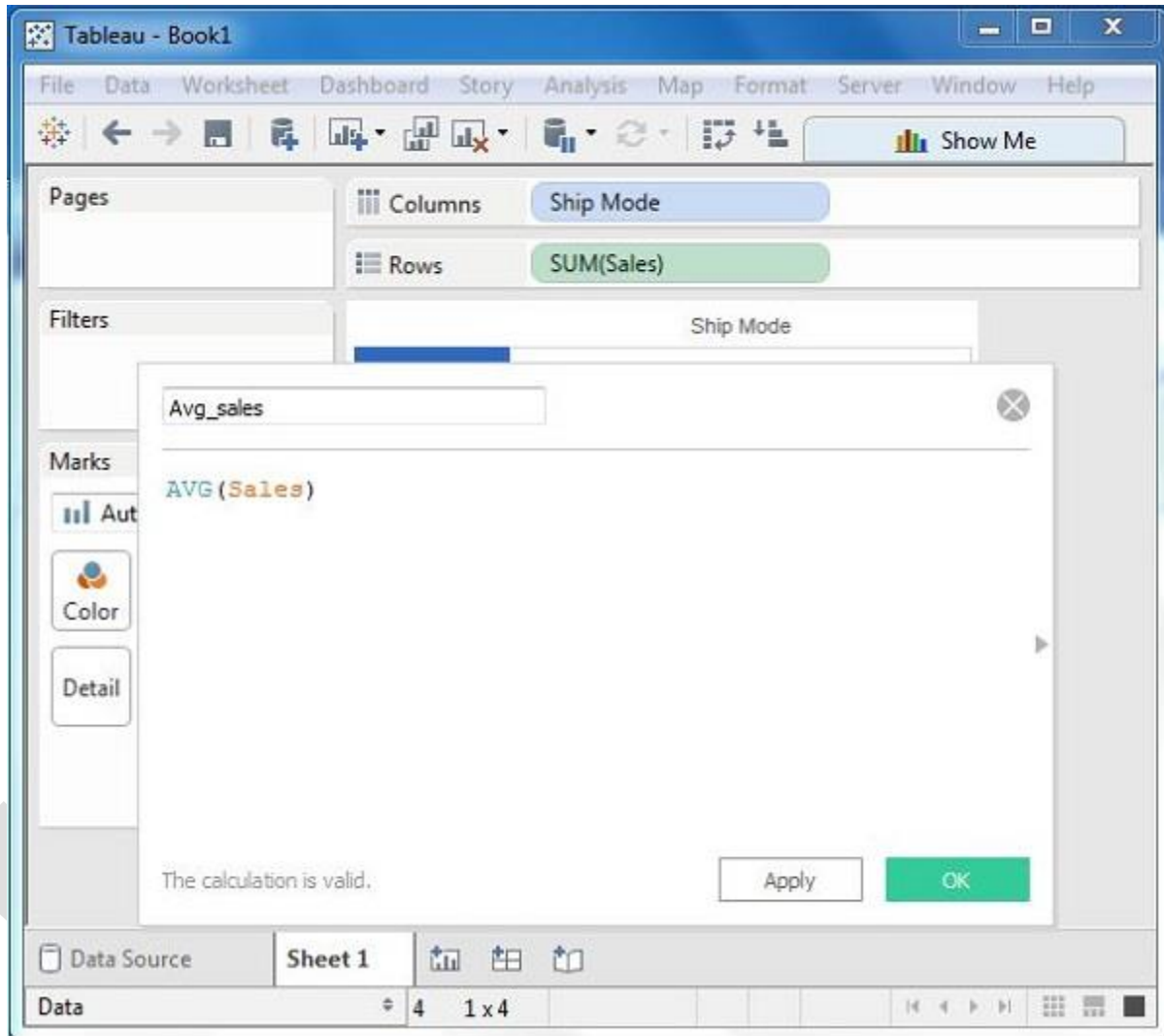
Using the Calculated Field

The above calculated field can be used in the view by dragging it to the Rows shelf as shown in the following screenshot. It produces a bar chart showing the difference between profit and discount for different shipping modes.

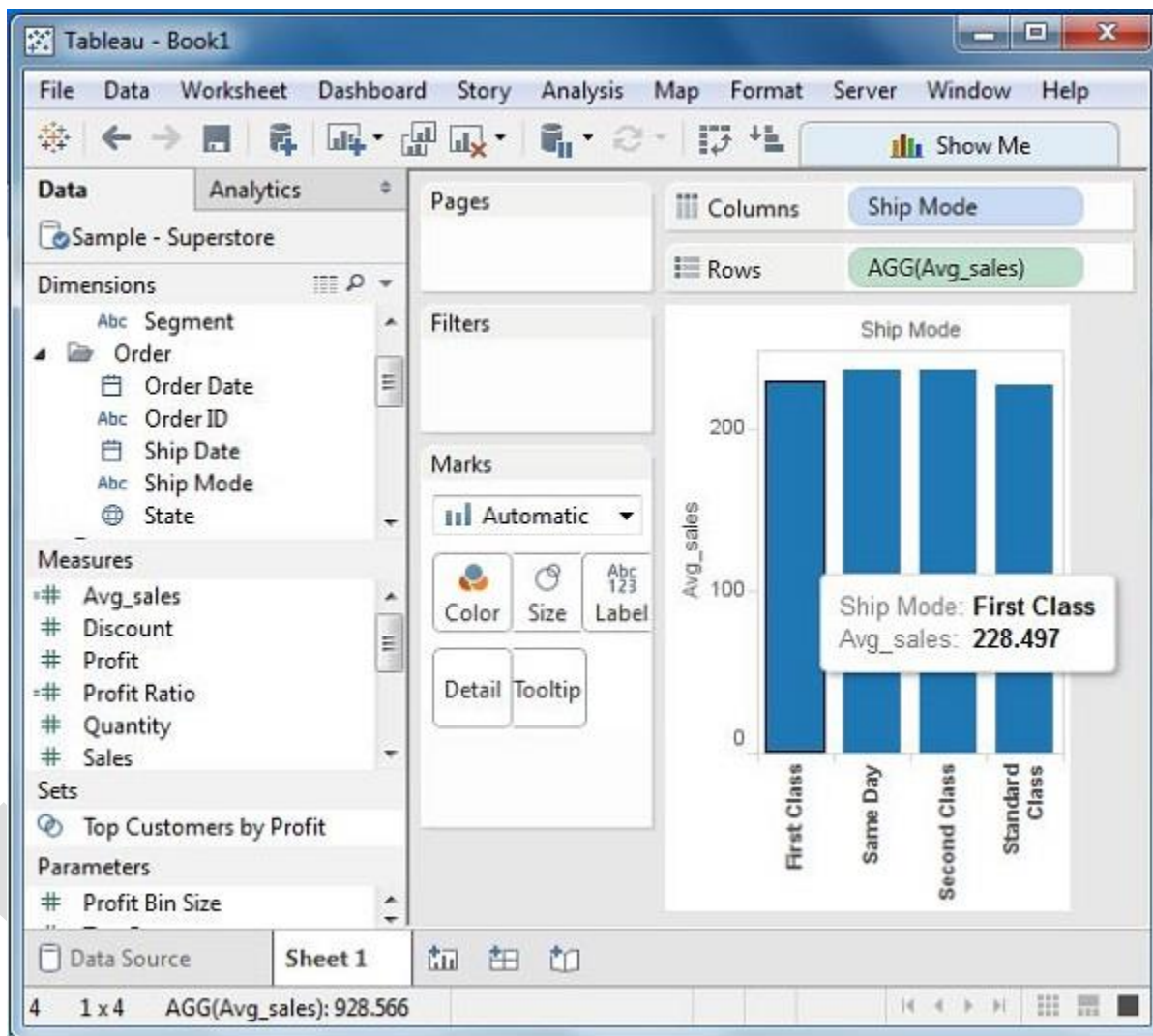


Applying Aggregate Calculations

In a similar manner as above, you can create a calculated field using aggregate function. Here, create AVG(sales) values for different ship mode. Write the formula in the calculation editor as shown in the following screenshot.



On clicking OK and dragging the Avg_Sales field to the Rows shelf, you will get the following view.



USING MAPS & DASHBOARDS and STORIES

Connect to your geographic data

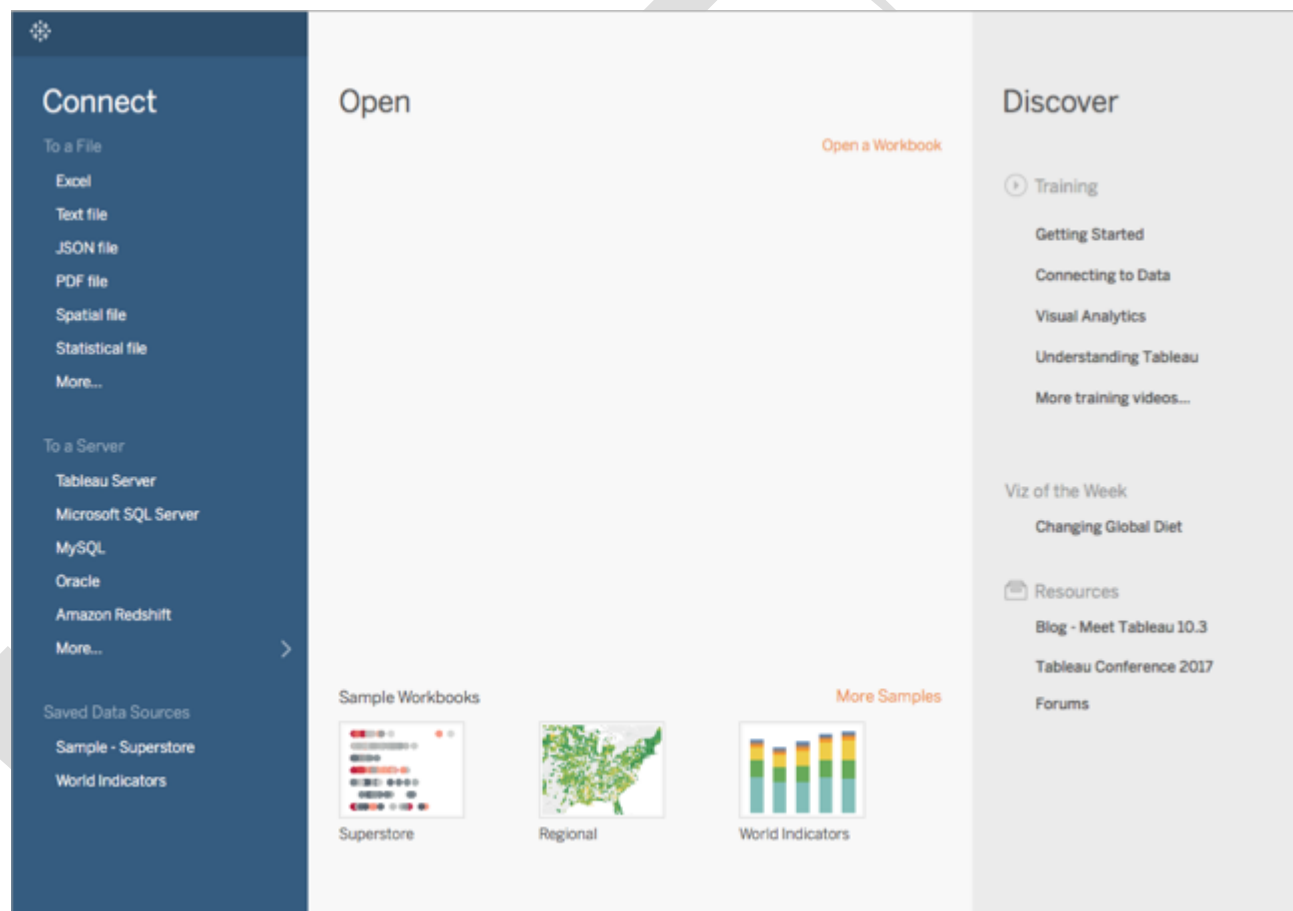
Geographic data comes in many shapes and formats. When you open Tableau Desktop, the start page shows you the connectors available in the left Connect pane. These are how you will connect to your data.

You can work with geographic data by connecting to spatial files, or you can connect to location data stored in spreadsheets, text files, or on a server.

Spatial files, such as a shapefile or geoJSON file, contain actual geometries (points, lines, or polygons), whereas text files or spreadsheets contain point locations in latitude and longitude

coordinates, or named locations that, when brought into Tableau, connect to the Tableau geocoding (stored geometries that your data references).

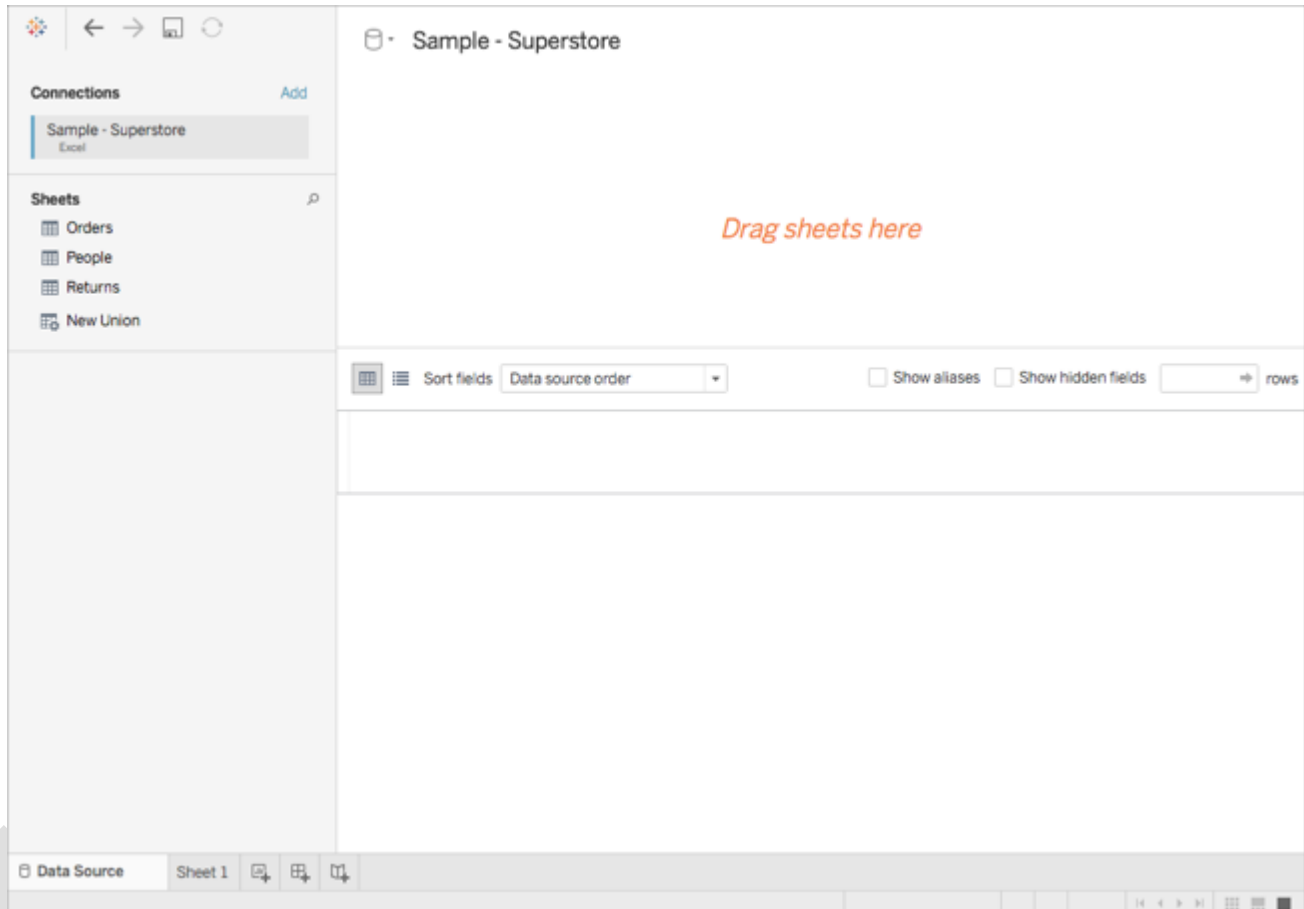
For a complete list of connections Tableau supports, see the list of [Data Connections](#) on the Tableau website.



For this tutorial, you are going to connect to an Excel file that comes with Tableau Desktop. It contains location names that Tableau can geocode. When you build a map view, the location names reference the geometries stored in the Tableau Map Service based on the geographic role you assign to the field. You'll learn more about geographic roles later in this tutorial.

1. Open Tableau Desktop.
2. In the Connect pane, click Excel.
3. Navigate to Documents > My Tableau Repository > Data Sources, and then open the Sample - Superstore.xls file.

Once you connect to the data source, your screen will look like this:



This is called the Data Source page, and it is where you can prepare your location data for use in Tableau.

Some of the tasks you can perform on the Data Source page include the following, but you don't have to do all these things to create a map view:

- Adding additional connections and joining your data
- Adding multiple sheets to your data source
- Assigning or changing geographic roles to your fields
- Changing the data type of your columns (from numbers to strings, for example)
- Renaming columns

- Splitting columns, such as splitting a full address into multiple columns for street, city, state, and postal code

For more information about the Data Source page and some of the tasks you can perform while on it, see the topics in the [Set Up Data Sources](#) section.

[Back to top](#)

Step 2: Join your data

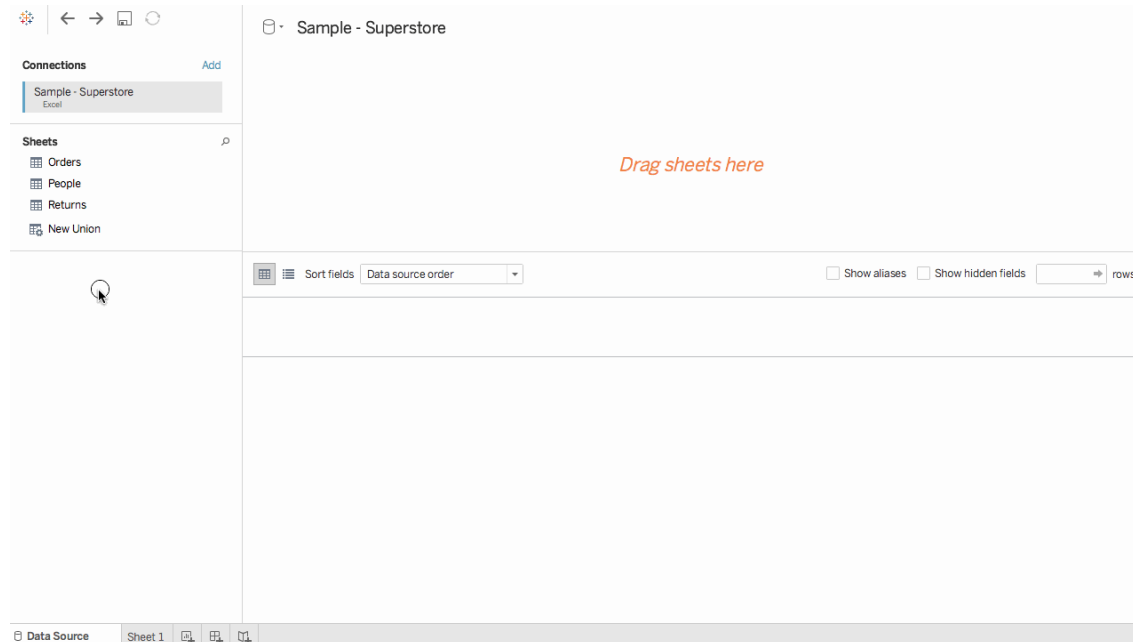
Your data is often held in multiple data sources or sheets. As long as those data sources or sheets have a column in common, you can join them in Tableau. Joining is a method for combining the related data on those common fields. The result of combining data using a join is a virtual table that is typically extended horizontally by adding columns of data.

Joining is often necessary with geographic data, particularly spatial data. For example, you can join a KML file that contains custom geographies for school districts in Oregon, U.S. with an Excel spreadsheet that contains demographic information about those school districts.

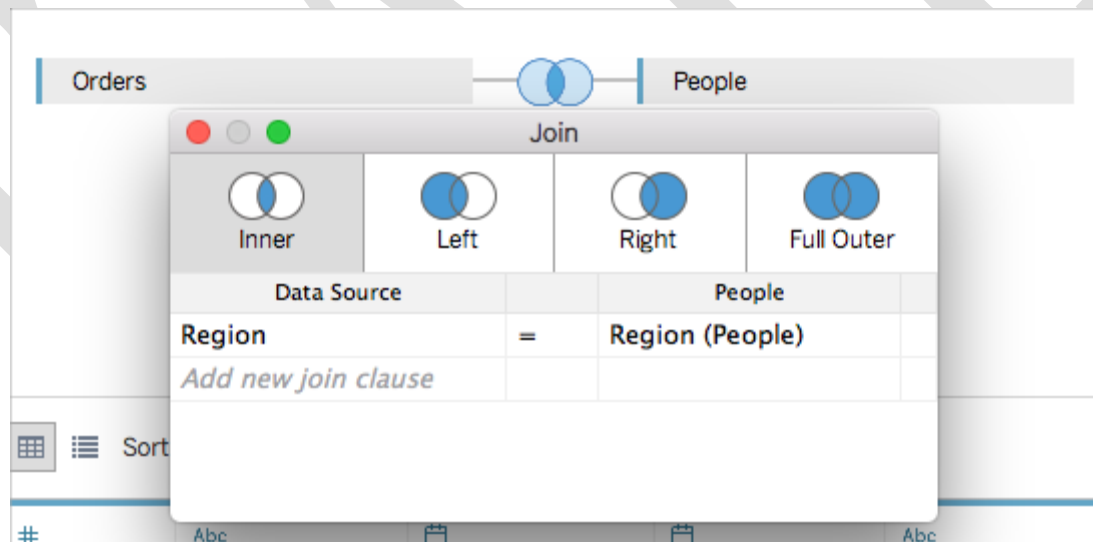
For this example, you will join two sheets in the Sample-Superstore data source.

1. On the left side of the Data Source page, under Sheets, double-click Orders.
2. Under Sheets, double-click People.

Tableau creates an inner-join between the two spreadsheets, using the Region column from both spreadsheets as the joining field. Now there is a sales person assigned to every location in your data source, as well as to regions.



To edit this join, click the join icon (the two circles). You can edit the join in the Join dialog box that opens. For more information about joining data in Tableau, see [Join Your Data](#).



[Back to top](#)

Step 3: Format your geographic data in Tableau

After you set up your data source, you might need to prepare your geographic data for use in Tableau. Not all of these procedures will always be necessary to create a map view, but it's important information to know when it comes to preparing geographic data for use in Tableau.

Depending on the type of map you want to create, you must assign certain data types, data roles, and geographic roles to your fields (or columns).

For example, in most cases, your latitude and longitude fields should have a *data type* of number (decimal), a *data role* of measure, and be assigned the Latitude and Longitude *geographic roles*. All other geographic fields should have a *data type* of string, a *data role* of dimension, and be assigned the appropriate geographic roles.

Note: If you are connecting to a spatial file, a Geometry field is created. It should have a data role of measure.




This step demonstrates how to format your geographic data to meet this criteria.

Change the data type of a column

When you first connect to geographic data, Tableau assigns data types to all of your columns. These data types include Number (decimal), Number (whole), Date and Time, Date, String, and Boolean. Sometimes Tableau does not get these data types right, and you must edit them. For example, Tableau might assign a Postal Code column a data type of Number (whole). To create map views, your Postal Code data must have a data type of String.

To change the data type of a column:

1. On the Data Source page, click the data type icon (the globe) for Postal Code and select String.

 Orders State	 Orders Postal Code	Abc Orders Region	 Abc Orders Product ID
Kentucky	42420	South	FUR-BS-10001798
Kentucky	42420	South	FUR-CH-10000454
California	90036	West	OFF-LA-10000240
Florida	33311	South	FUR-TA-10000577
Florida	33311	South	OFF-ST-10000760
California	90032	West	FUR-FU-10001487
California	90032	West	OFF-AR-10002833
California	90032	West	TEC-PH-10002275
California	90032	West	OFF-BI-10003910
California	90032	West	OFF-AP-10002892

For more information about data types, see [Data Types](#).

Assign geographic roles to your geographic data

In Tableau, a *geographic role* associates each value in a field with a latitude and longitude value. When you assign the correct geographic role to a field, Tableau assigns latitude and longitude values to each location in that field by finding a match that is already built in to the installed geocoding database. This is how Tableau knows where to plot your locations on the map.

When you assign a geographic role to a field, such as State, Tableau creates a Latitude (generated) field and a Longitude (generated) field.

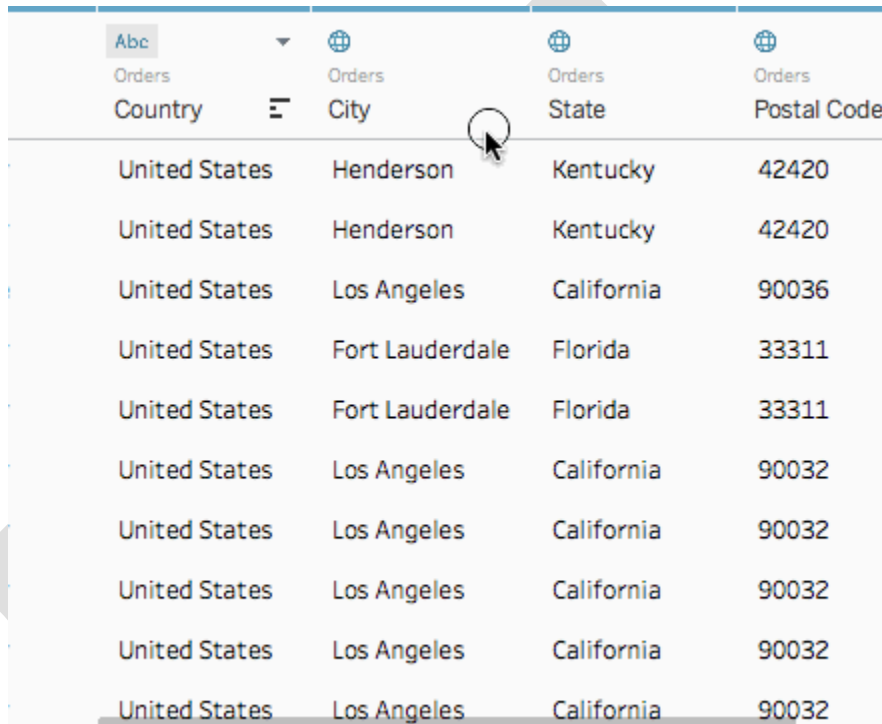
Geographic roles are sometimes automatically assigned to your data, such as in this example. You can tell a geographic role has been assigned to your data because the column includes a globe icon.

If a geographic role is not automatically assigned, you can manually assign one to your field. You don't need to do so for this example, but it's important to know how so you can do it for your own data.

To assign or edit a geographic role:

1. On the Data Source page, click the globe icon.
2. Select Geographic Role, and then select a role that best matches your data.

For example, in this case, the Country column does not have a geographic role assigned to it, so the Country/Region geographic role is assigned.



Country	City	State	Postal Code
United States	Henderson	Kentucky	42420
United States	Henderson	Kentucky	42420
United States	Los Angeles	California	90036
United States	Fort Lauderdale	Florida	33311
United States	Fort Lauderdale	Florida	33311
United States	Los Angeles	California	90032
United States	Los Angeles	California	90032
United States	Los Angeles	California	90032
United States	Los Angeles	California	90032
United States	Los Angeles	California	90032

Note: If you have difficulties assigning geographic roles to your data, or have data that is not built in to the Tableau map server, there are a few things you can do to get that data into Tableau. See [Assign Geographic Roles](#).

Change from dimensions to measures

When you connect to geographic data, Tableau also assigns data roles to all of your columns. A column can be a *dimension* or *measure*. In most cases, your latitude and longitude columns should be measures. For special cases, such as if you want to plot every location in your data source on a map without the ability to drill up or down a level of detail (such as from City to State), they can be dimensions. A great example of this is a [point distribution](#) map.

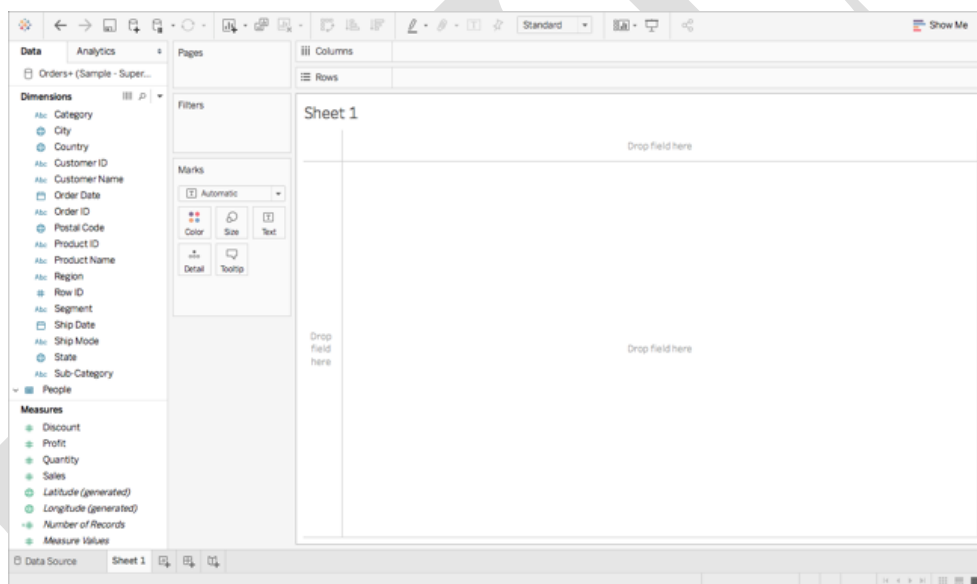
The rest of your geographic data should be dimensions.

You don't need to change the data role of a column for this example, but it's important to know how so you can do it for your own data. Feel free to practice here. You can always undo any changes you make.

To change the data role of a column:

1. On the Data Source page, click Sheet 1.

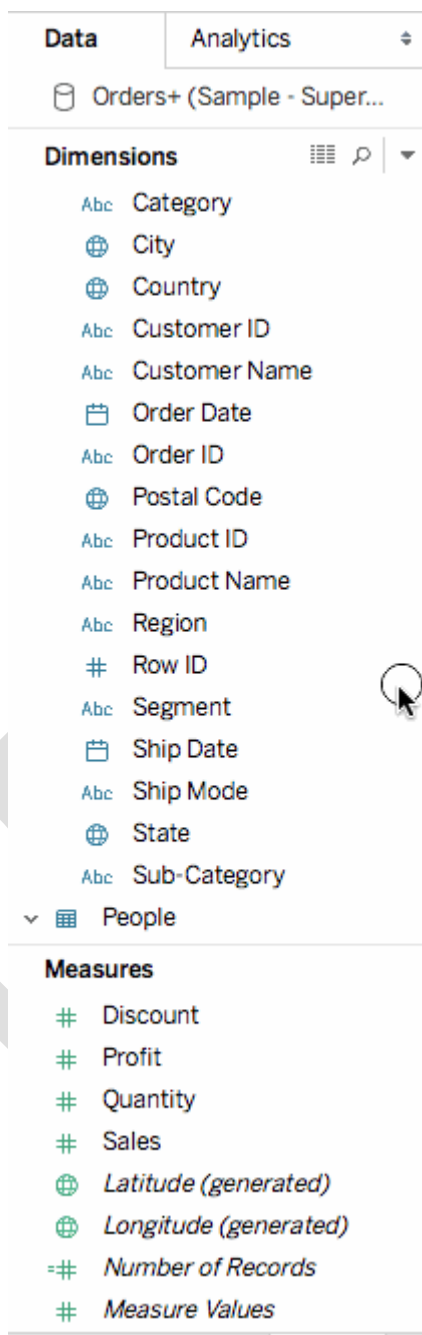
Your workspace updates to look like this:



This is called a worksheet, and it is where you will build your map. On the left-side of the screen is the Data pane. All of the columns in your data source are listed as fields in this pane. For example, Country and State. These fields contain all the raw data in your columns. Note that Tableau has generated a Latitude and Longitude field (*Latitude (generated)* and *Longitude (generated)*). This is because you assigned geographic roles to your data.

The fields in the data pane are divided into measures and dimensions. The fields placed in the Dimensions section of the Data pane are often categorical data, such as Date and Customer ID, while the fields placed in the Measures section of the Data pane are often quantitative data, such as Sales and Quantity.

2. In the Data pane, under Dimensions, select a field, such as Row ID, and drag it down to the Measures section.



The screenshot shows a data analytics tool interface. At the top, there are tabs for 'Data' and 'Analytics'. Below the tabs, a sample data source 'Orders+ (Sample - Super...)' is selected. The main area is divided into two sections: 'Dimensions' and 'Measures'. The 'Dimensions' section contains a list of fields: 'Category', 'City', 'Country', 'Customer ID', 'Customer Name', 'Order Date', 'Order ID', 'Postal Code', 'Product ID', 'Product Name', 'Region', 'Row ID', 'Segment', 'Ship Date', 'Ship Mode', 'State', and 'Sub-Category'. The 'Measures' section contains a list of fields: 'Discount', 'Profit', 'Quantity', 'Sales', 'Latitude (generated)', 'Longitude (generated)', 'Number of Records', and 'Measure Values'. A mouse cursor is hovering over the 'Row ID' field in the Dimensions list.

The field is added to the Measures section and changes from blue to green. You just converted a Dimension to a Measure. To convert a field from a measure to a dimension, drag the field from the Measures section up to the Dimensions section.

For more information, see [Dimensions and Measures, Blue and Green](#).

Step 4: Create a geographic hierarchy

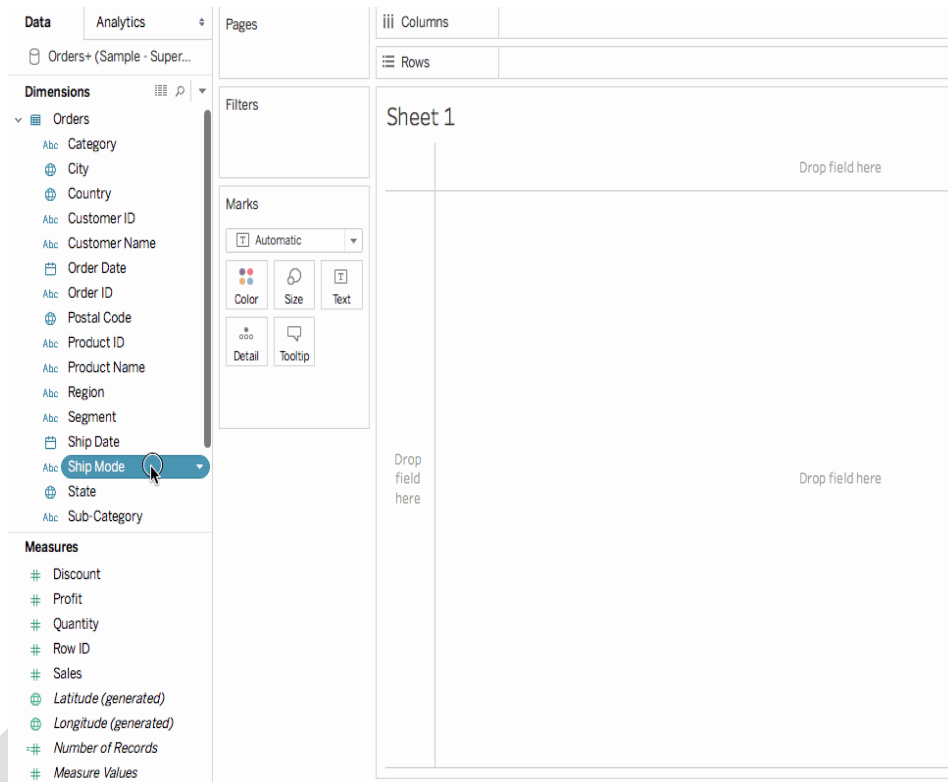
Now that you are in the worksheet space, you can create geographic hierarchies. This is not required to create a map view, but creating a geographic hierarchy will allow you to quickly drill into the levels of geographic detail your data contains, in the order you specify.

To create a geographic hierarchy:

1. In the Data pane, right-click the geographic field, Country, and then select Hierarchy > Create Hierarchy.
2. In the Create Hierarchy dialog box that opens, give the hierarchy a name, such as Mapping Items, and then click OK.

At the bottom of the Dimensions section, the Mapping Items hierarchy is created with the Country field.

3. In the Data pane, drag the State field to the hierarchy and place it below the Country field.
4. Repeat step 3 for the City and Postal Code fields.



When you are finished, your hierarchy should be in the following order:

- Country
- State
- City
- Postal Code

Step 5: Build a basic map

Now that you have connected to and joined your data, formatted your data, and built a geographic hierarchy, you are now ready to start building your map. You will start by building a basic map view.

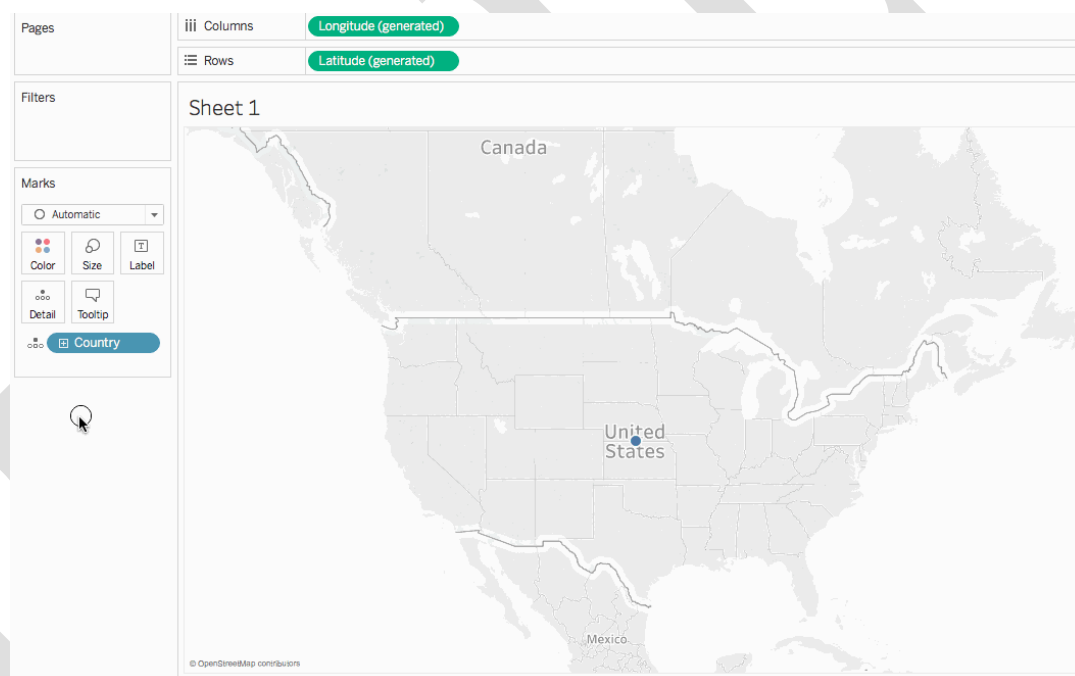
1. In the Data pane, double-click Country.

The Country field is added to Detail on the Marks card, and Latitude (generated) and Longitude (generated) are added to the Columns and Rows shelves. A map view with one data point is created. Since a geographic role is assigned to Country, Tableau creates a

map view. If you double-click any other field, such as a dimension or measure, Tableau adds that field to the Rows or Columns shelf, or the Marks card, depending on what you already have in the view. Geographic fields are always placed on Detail on the Marks card, however.

Since this data source only contains one country, (United States), that is the only data point shown. You will need to add more levels of detail to see additional data points. Since you created a geographic hierarchy, this is easy.

2. On the Marks card, click the + icon on the Country field.



The State field is added to Detail on the Marks card and the map updates to include a data point for every state in the data source.

If you did not create a hierarchy, the + icon on the Country field will not be available. In this case, to add State as another level of detail, manually drag State from the Data pane to Detail on the Marks card.

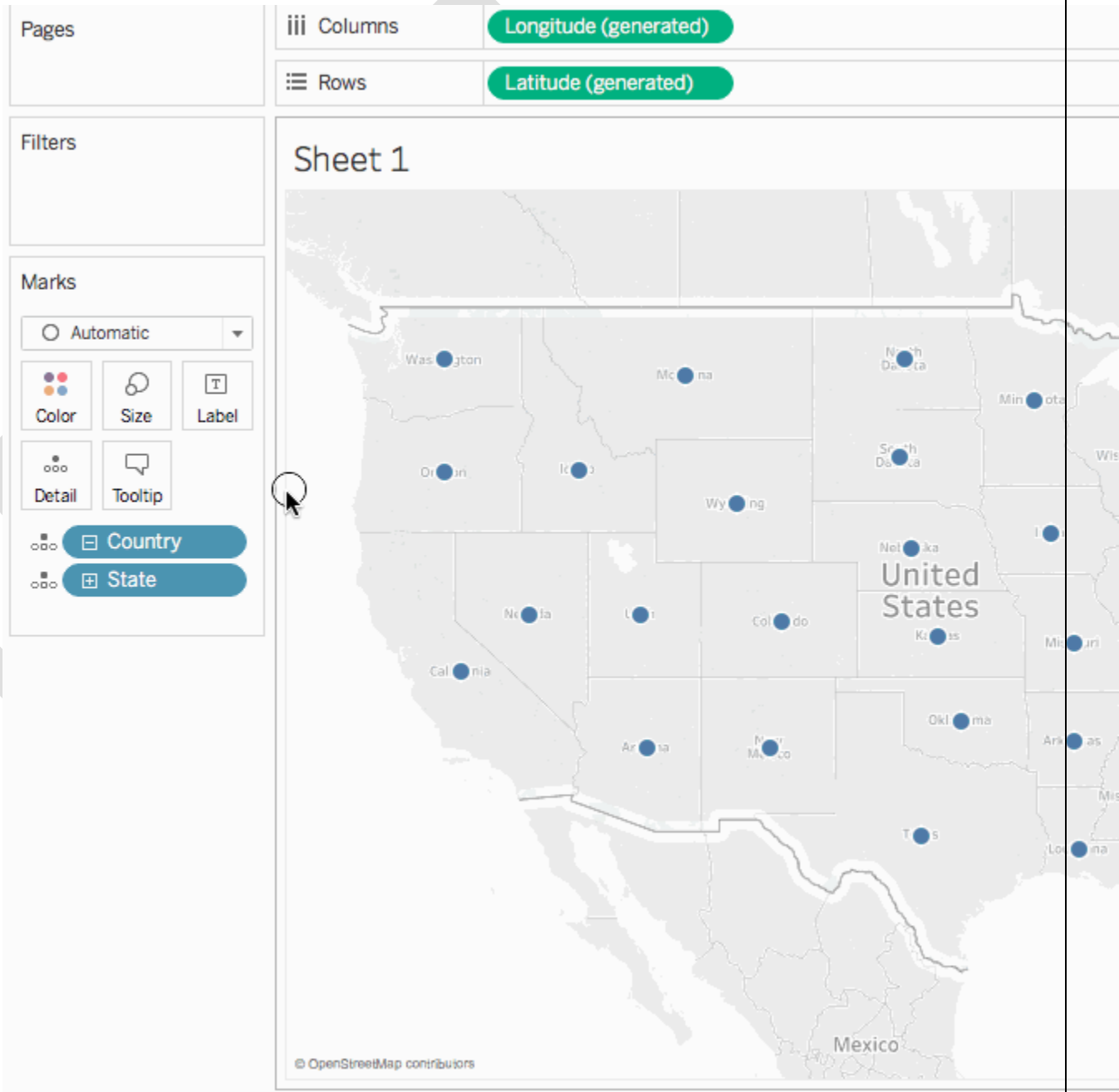
Congratulations! You now have a basic map view that you can customize and build upon in the next steps.

Step 6: Change from points to polygons

The default map type in Tableau is often a point map. When you have geographic roles assigned to your geographic data, however, it's easy to change those data points to polygons.

Note: Filled maps are not available for cities or airports.

1. On the Marks card, click the Mark Type drop-down and select Filled Map.



The map updates to a polygon map.

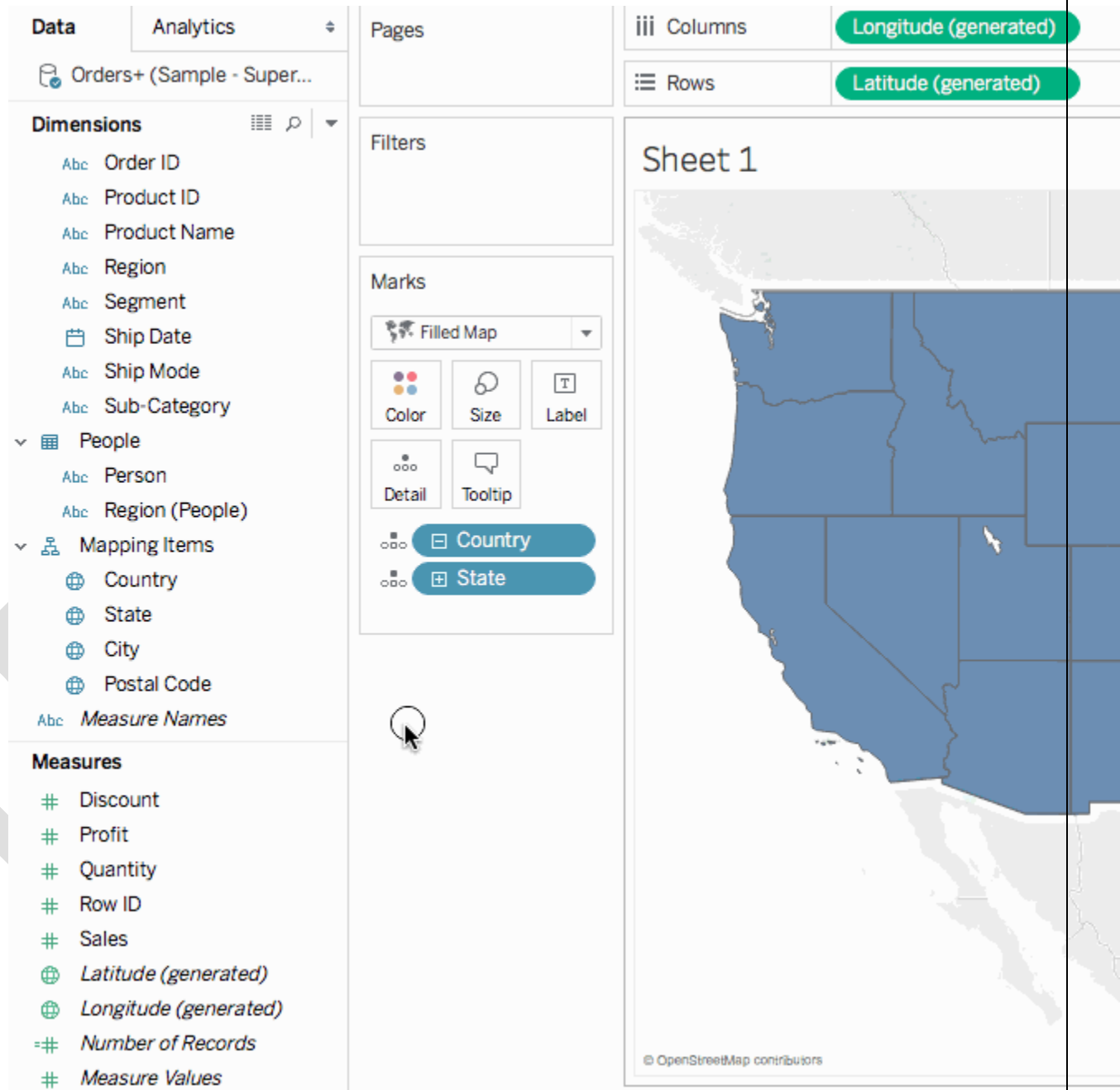
[Back to top](#)

Step 7: Add visual detail

You can add measures and dimensions to the Marks card to add visual detail to your view. In this example, you will add color and labels to the view.

Add color

- From Measures, drag Sales to Color on the Marks card.



Each state is colored by sum of sales. Since Sales is a measure, a qualitative color palette is used. If you place a dimension on color, then a categorical color palette is used.

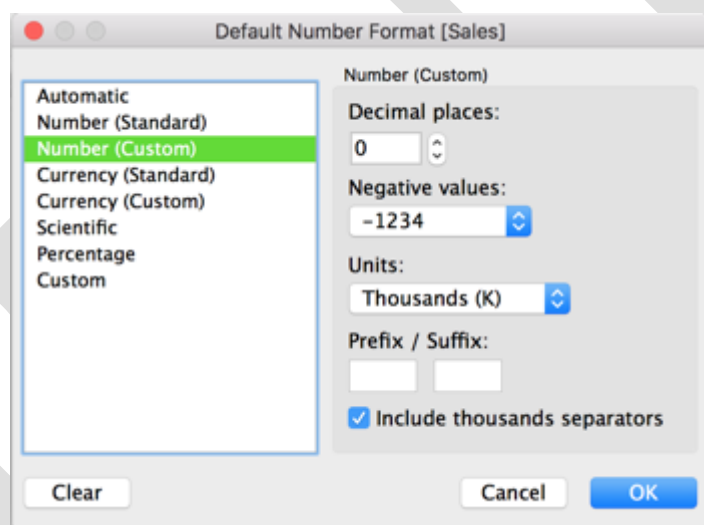
Add labels

1. From Measures, drag Sales to Label on the Marks card.

Each state is labeled with sum of sales. The numbers need a little bit of formatting, however.

2. In the Data pane, right-click Sales and select Default Properties > Number Format.
3. In the Default Number Format dialog box that opens, select Number (Custom), and then do the following:
 - For Decimal Places, enter 0.
 - For Units, select Thousands (K).
 - Click OK.

The labels and the color legend update with the specified format.



[Back to top](#)

Step 8: Customize your background map

The background map is everything behind your marks (borders, oceans, location names, etc.) You can customize the style of this background map, as well as add map layers and data layers. In addition to customizing the background maps, you can also connect to your own WMS server or Mapbox map. For more information, see [Use Web Map Service \(WMS\) Servers](#) and [Use Mapbox Maps](#).

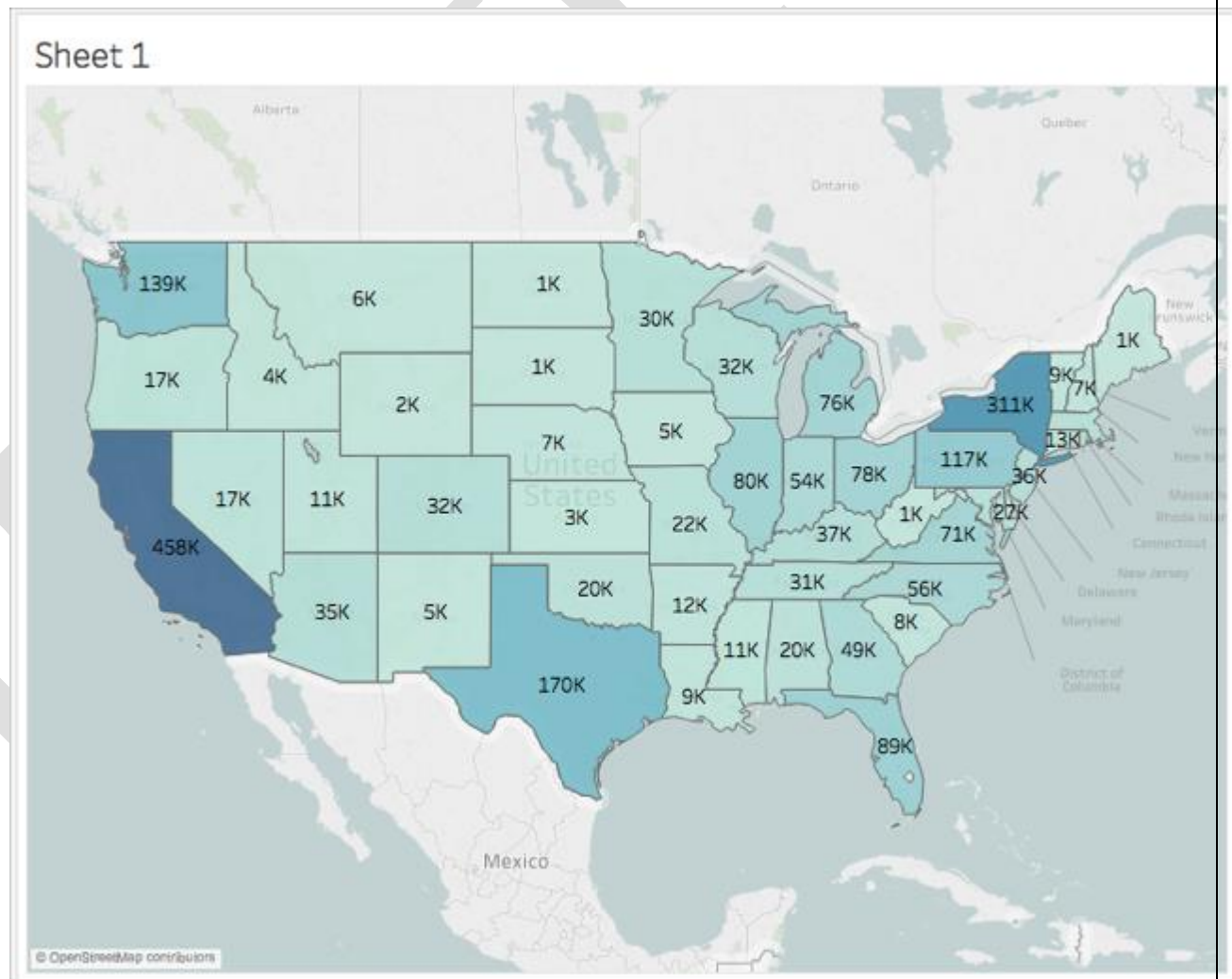
To customize your background map:

1. Select Map > Map Layers.

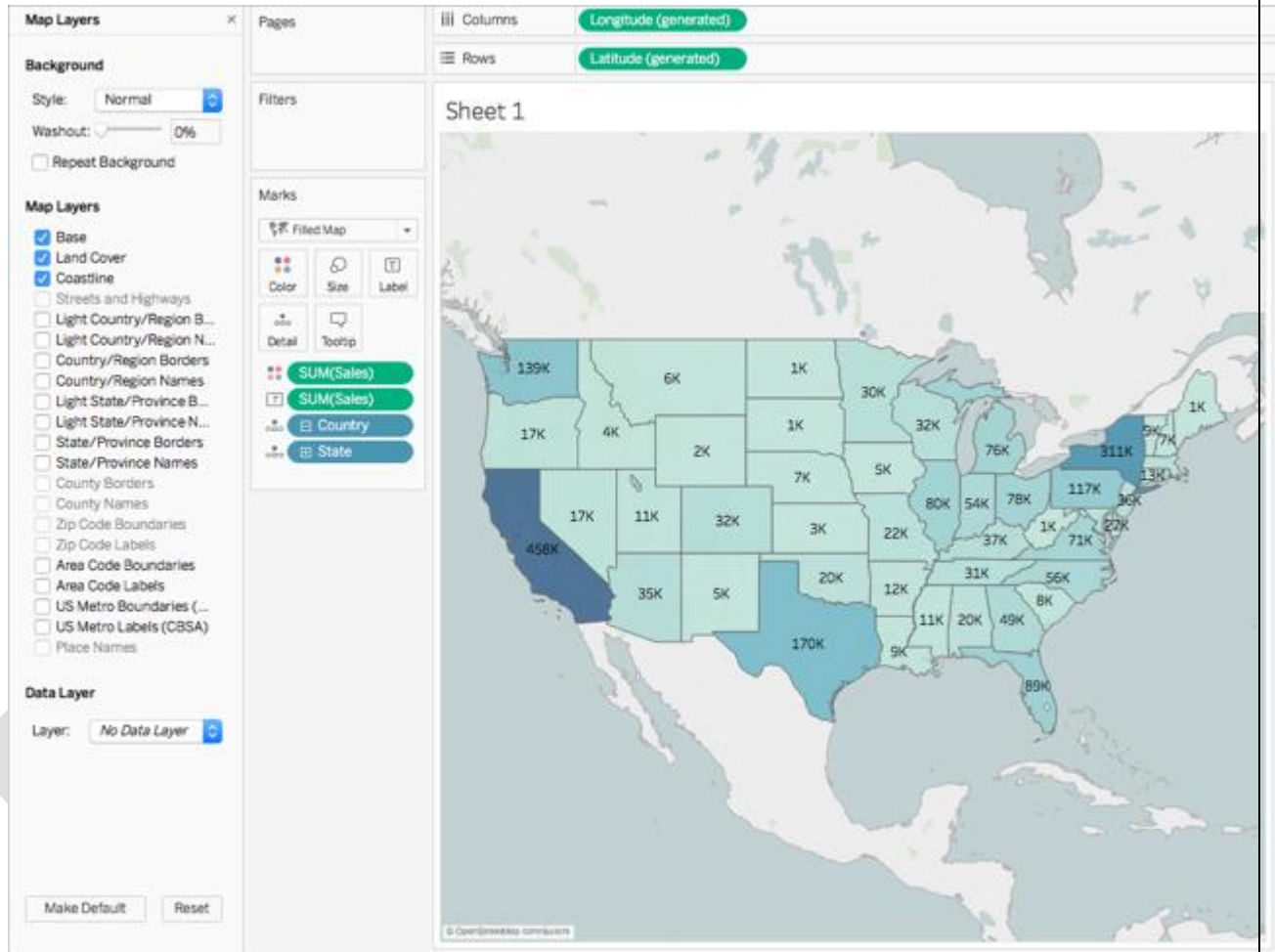
The Map Layers pane appears on the left side of the workspace. This is where all background map customization happens.

2. In the Map Layers pane, click the Style drop-down and select Normal.

The background map updates to look like this:



3. In the Map Layers pane, under Map Layers, select Coastlines, and then clear Country/Region Borders, Country/Region Names, State/Province Borders, and State/Province Names.



4. At the top of the Map Layers pane, click the X to return to the Data pane.

The background map is now simplified to draw attention to your data.

[Back to top](#)

Step 9: Create custom territories

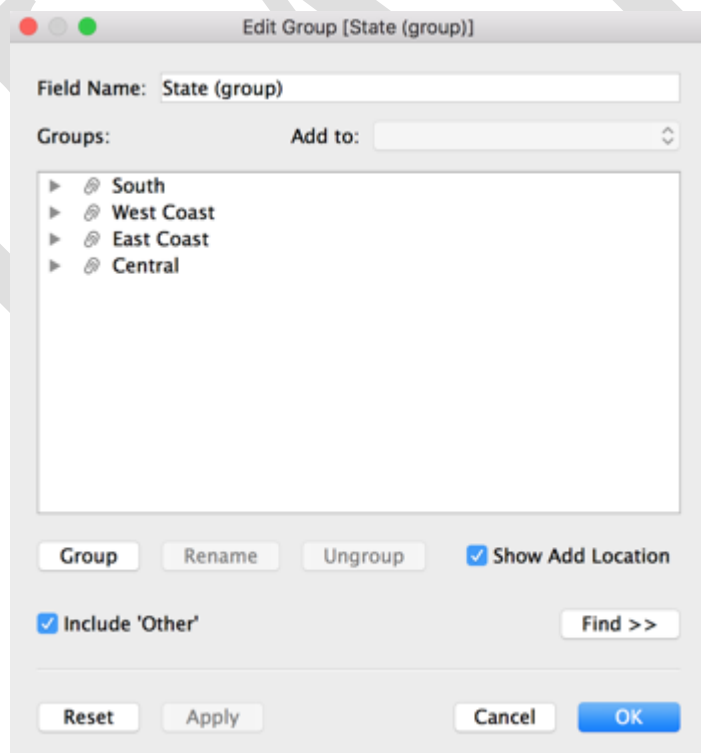
As you build your map view, you might want to group existing locations together to create your own territories or regions, such as sales territories for your organization.

1. In the Data pane, right-click State and select Create > Group.

2. In the Create Group dialog box that opens, select California, Oregon, and Washington , and then click Group. Each group you create represents a territory.

Note: To multi-select, hold down Ctrl (Command on Mac) as you select states.

3. Right-click the new group you just created and select Rename.
4. Rename the group, West Coast.
5. For the next territory, select Alabama, Florida, Georgia, Louisiana, Mississippi, South Carolina, and Texas, and then click Group.
6. Rename this group, South.
7. For the third territory, select Connecticut, Delaware, District of Columbia, Main, Maryland, Massachusetts, New Hampshire, New Jersey, New York, Pennsylvania, Rhode Island, Vermont, and finally, West Virginia, and then click Group.
8. Rename this group, East Coast.
9. Select Include Other to group the remaining states.
10. Rename the Other group, Central.

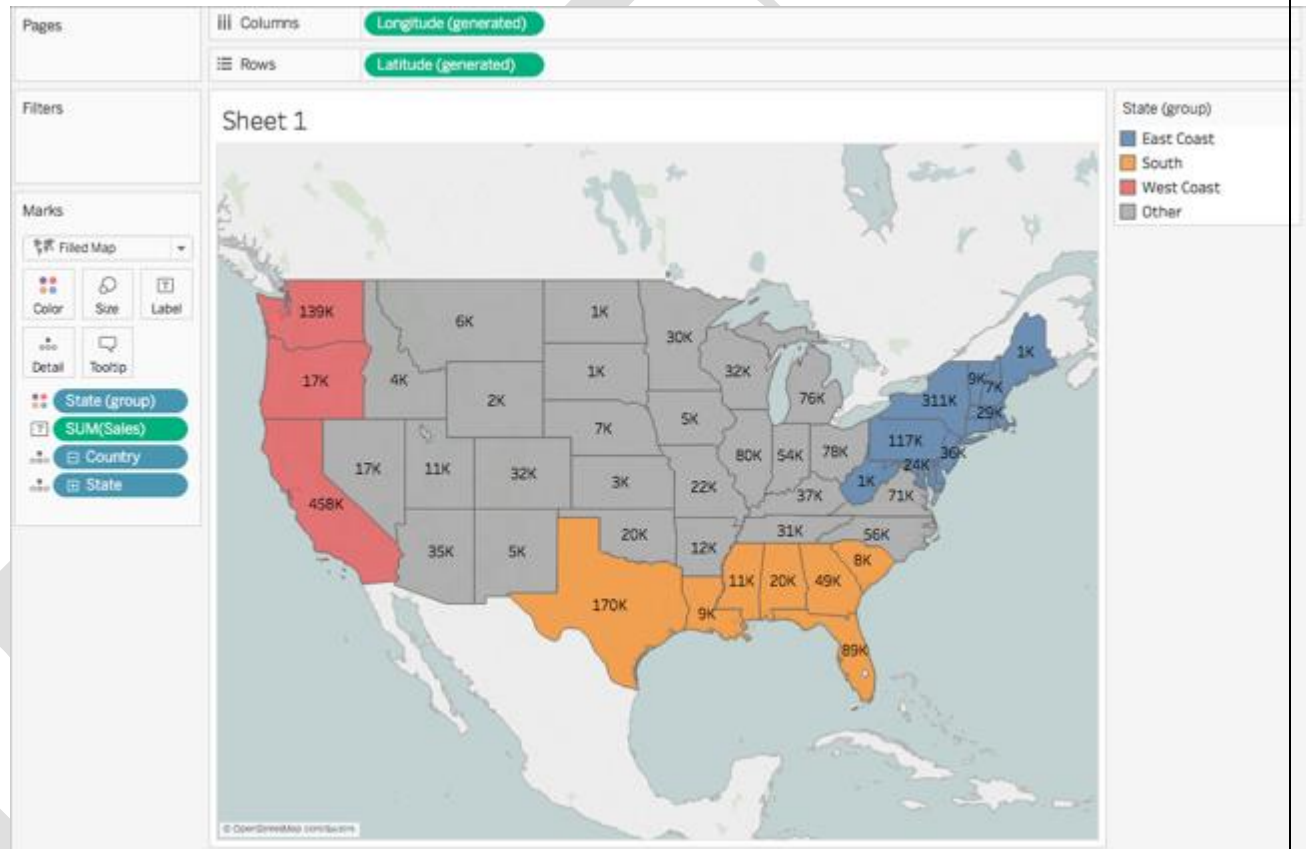


11. Click OK.

A State (group) field appears in the Data pane beneath your other mapping items.

12. From the Data pane, drag State (group) to Color on the Marks card.

The view updates to look like this:

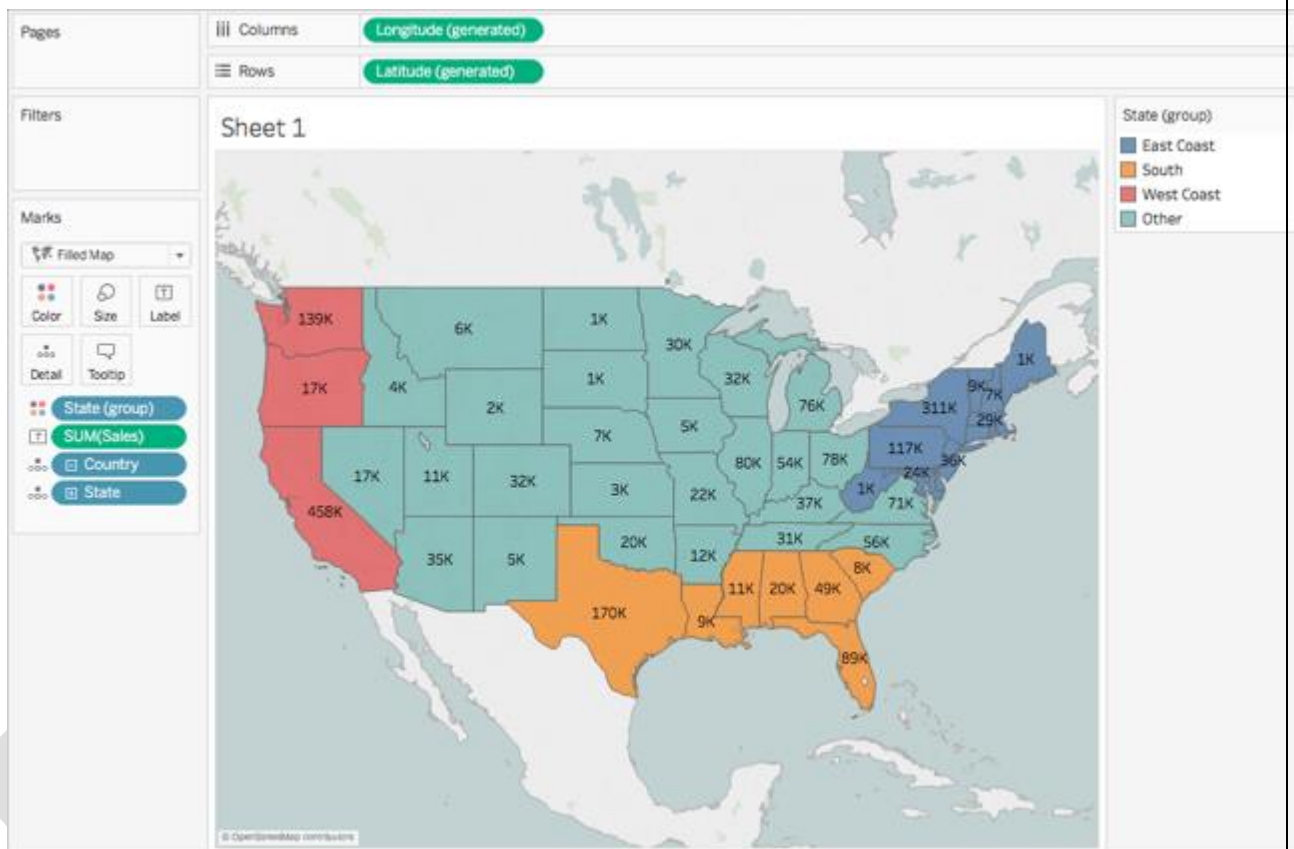


Notice that each group has a different color.

13. On the Marks card, click the Color icon and select Edit Colors.

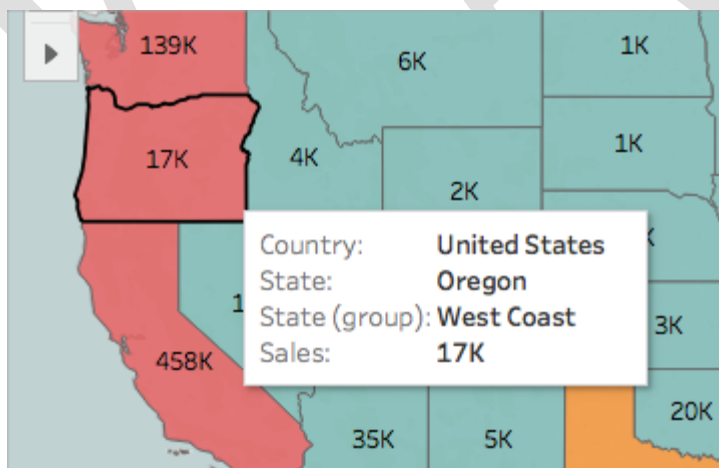
14. In the Edit Colors dialog box that appears, select Assign Palette, and then click OK.

The marks update with new colors.



15. From Measures, drag Sales to Tooltip on the Marks card.

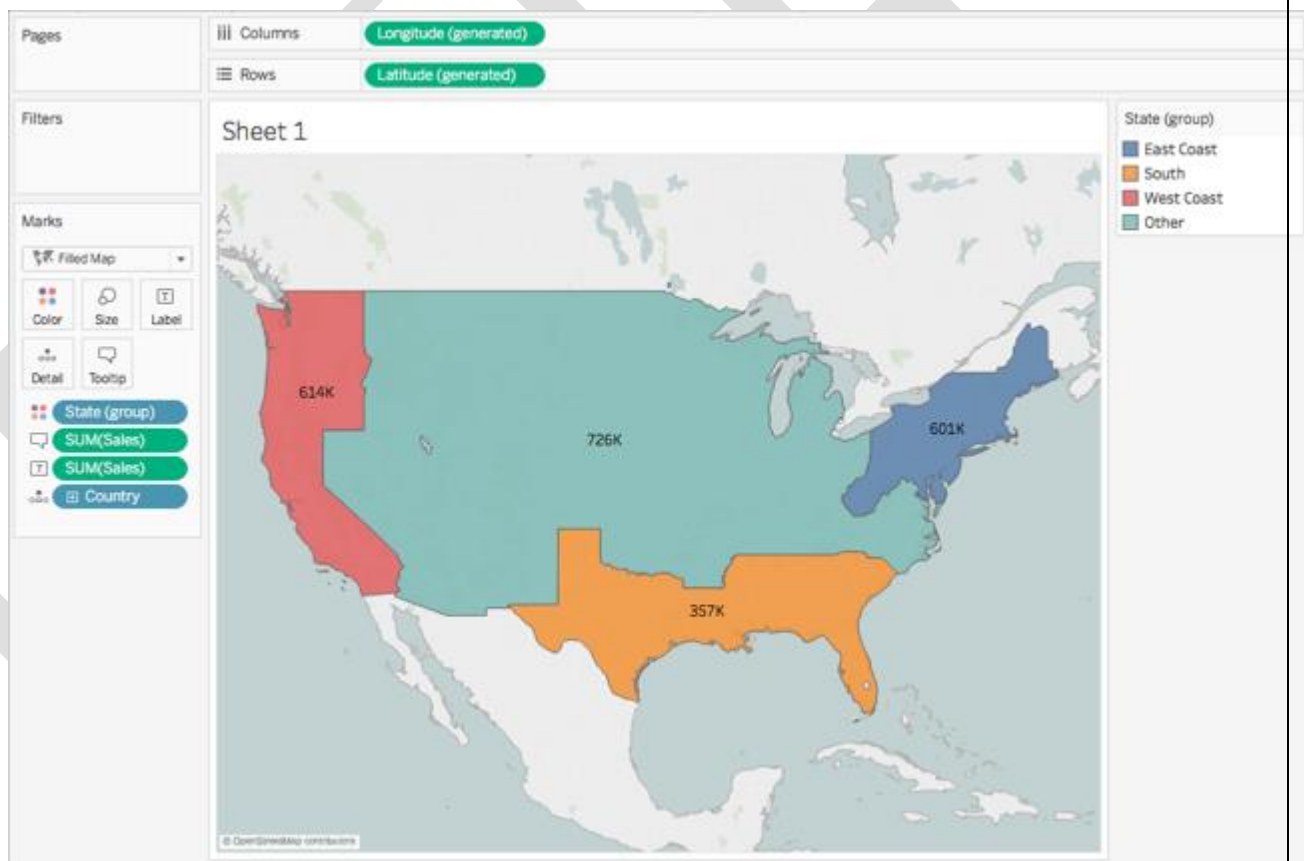
When you hover over a state, a tooltip appears with the sales for that state, among other information. You'll learn how to edit this tooltip later.



16. On the Marks card, click the minus (-) icon on the Country field to remove State from the level of detail.

If you did not create a hierarchy, you can drag State from the view to remove it. You can remove any field by dragging it from the view.

The states no longer appear on the map. Notice how the sum of sales has updated for the labels and in the tooltip? This is because custom territories aggregate at the level of the group, rather than separately for each location within the group. So the sum of sales you are seeing in the West Coast group, for example, are the total sales for California, Oregon, and Washington combined.



[Back to top](#)

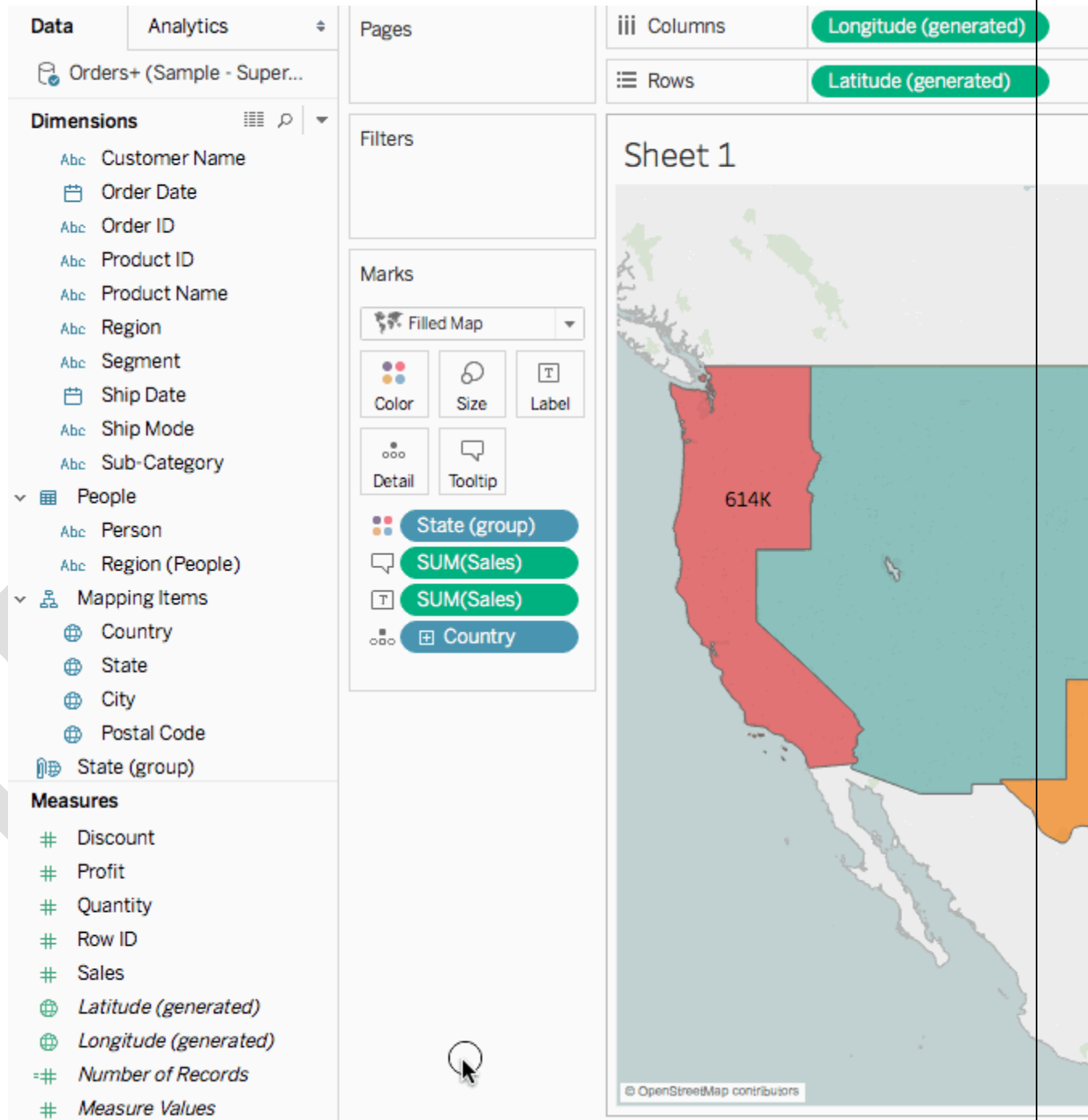
Step 10: Create a dual axis map

So far you have created two map views: one that shows the sales per state, and one that shows the sales per region. Could you layer these maps on top of one another? Yes! In Tableau, you can create a map with two layers of marks. This is called a dual axis map in Tableau, and is often used to layer points over polygons. In this example, you will layer two polygons maps.

To create a dual axis map:

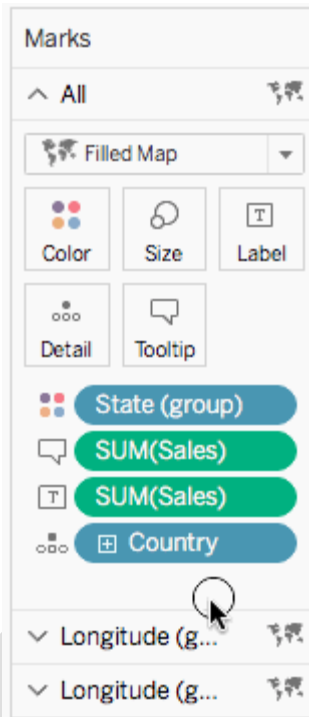
1. From the Data pane, drag Longitude (generated) to the Columns shelf and place it to the right of the first Longitude field.

The view updates with two identical maps.



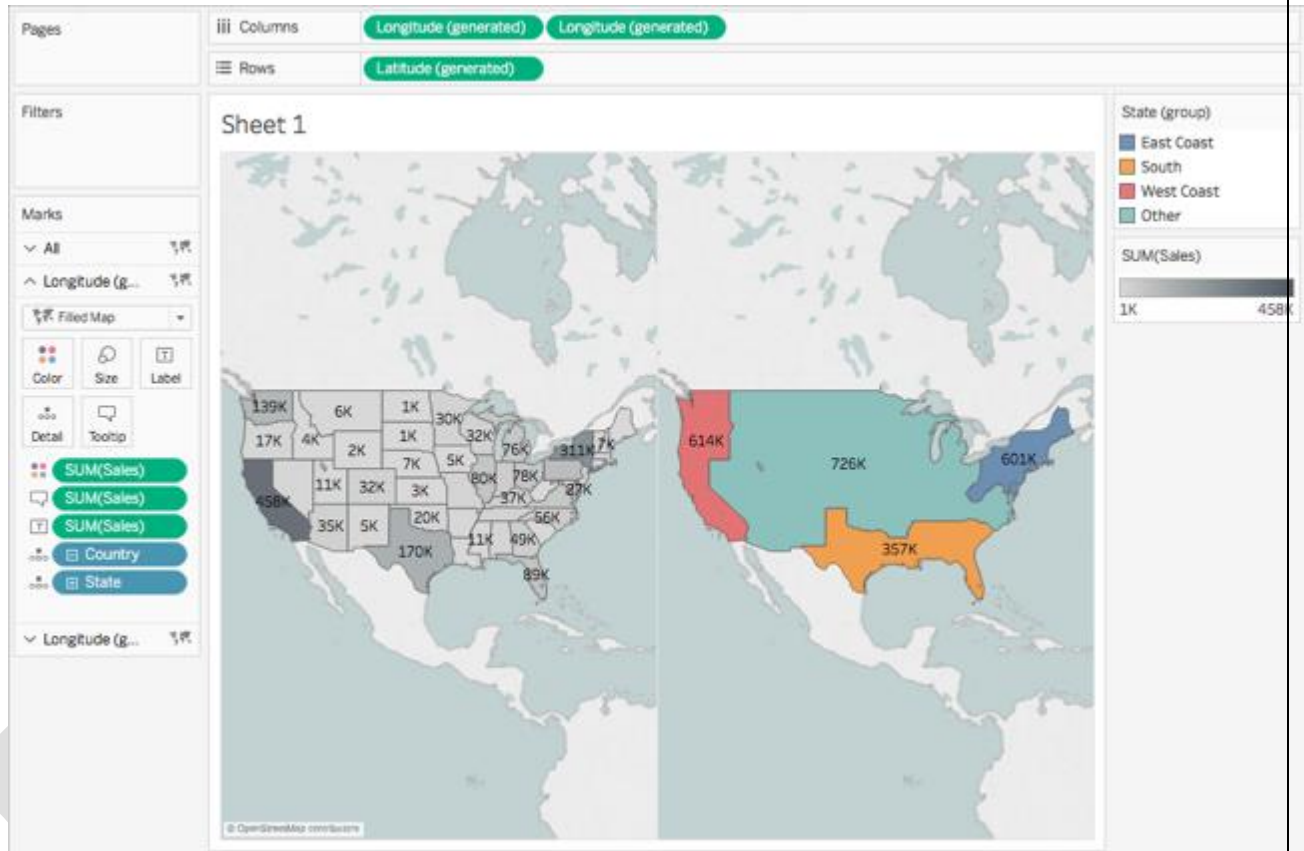
There are now three tabs on the Marks card: one for each map view, and one for both views (All). You can use these to control the visual detail of the map views. The top

Longitude tab corresponds to the map on the left of the view, and the bottom Longitude tab corresponds to the map on the right of the view.

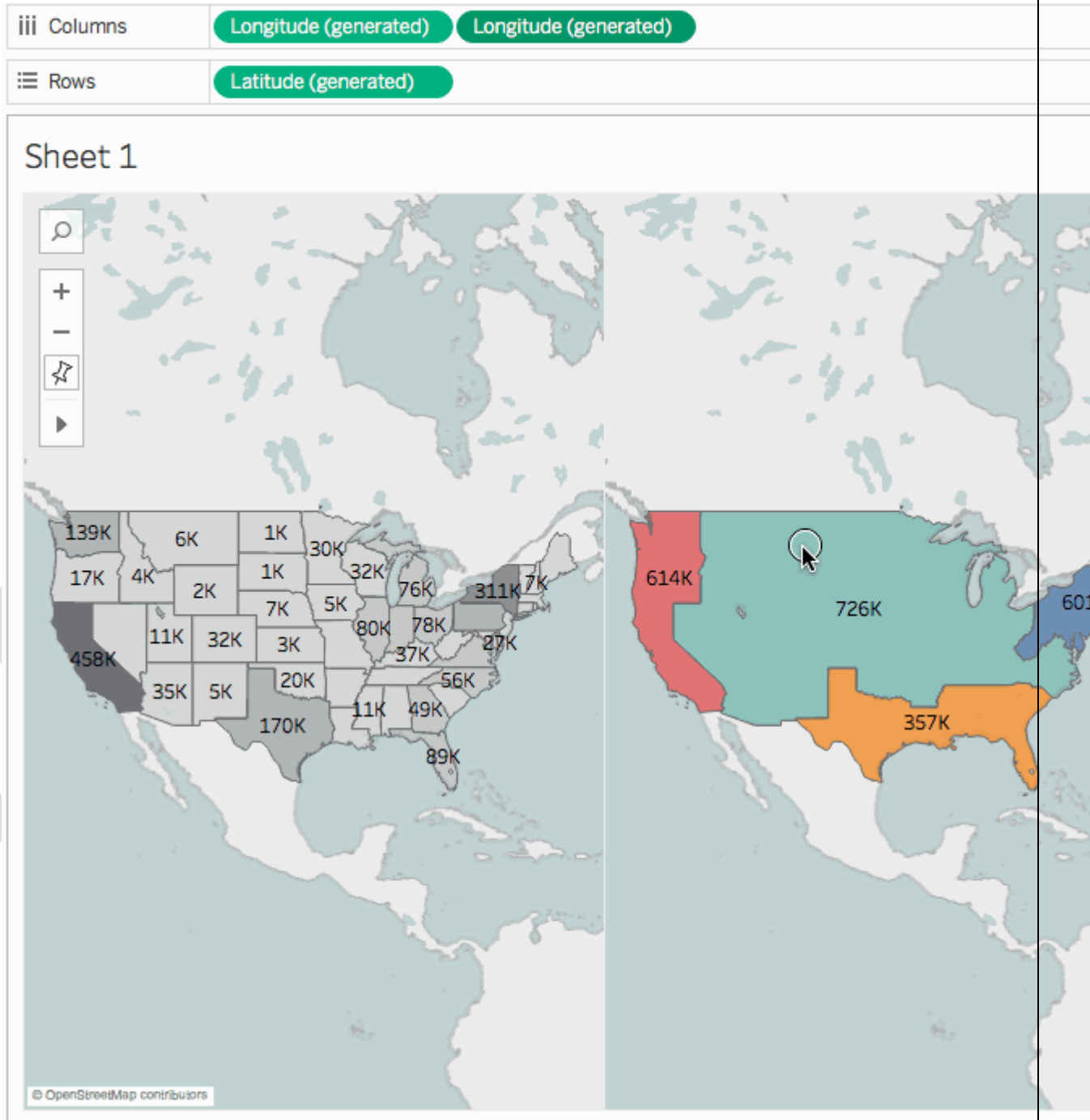


2. On the Marks card, select the top Longitude (generated) tab.
3. From Measures, drag Sales to Color on the top Longitude (generated) Marks card.
The map on the left updates.
4. On the top Longitude (generated) Marks card, click the + icon on the Country field to drill back down to the State level of detail.
5. On the Marks card, click Color, and then select Edit Colors.
6. In the Edit Colors dialog box that opens, click the Palette drop-down, select Gray, and then click OK.

At this point, your maps look like this:



7. On the Columns shelf, right-click the Longitude (generated) field on the right and select Dual Axis.



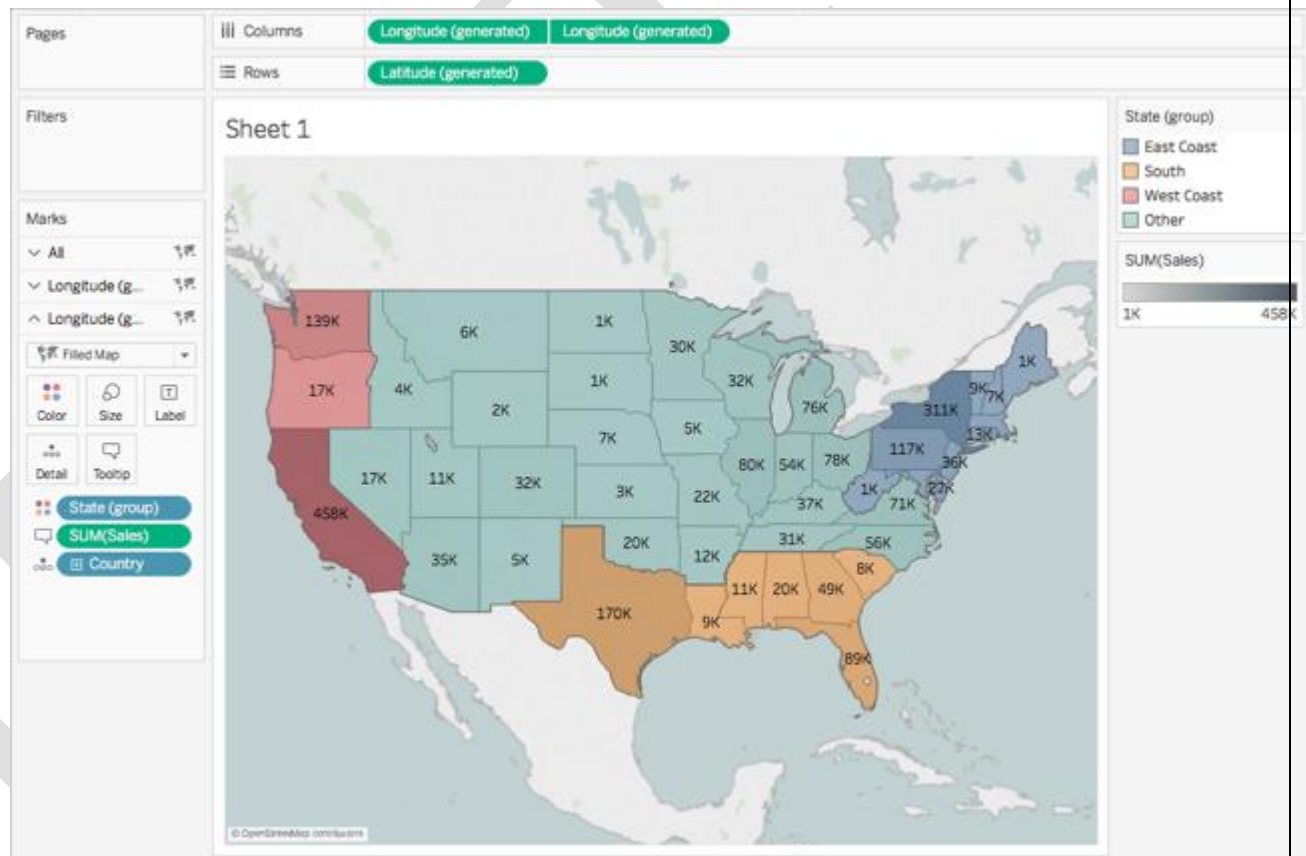
8. On the Marks card, select the bottom Longitude (generated) tab.
9. On the bottom Longitude (generated) Marks card, drag both SUM(Sales) fields from the view to remove them.

The labels for each map no longer overlap.

10. On the bottom Longitude (generated) Marks card, click Color, and then, for Opacity, enter 50%.

This is a crucial step if you want to be able to see the map on the bottom layer.

The map view updates to look like this:



You can now see how each state performed within each group.

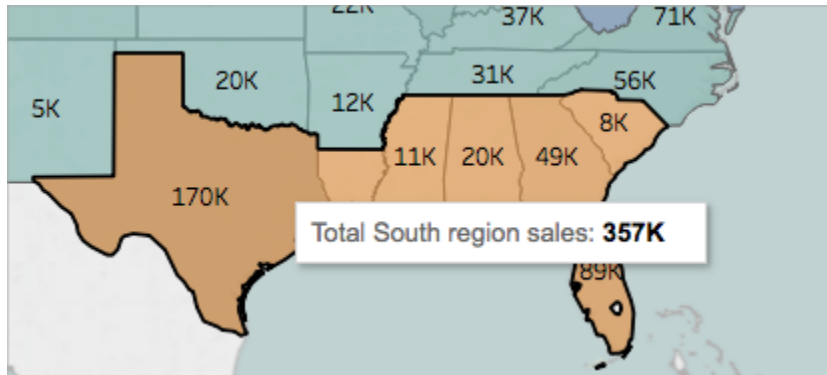
11. On the bottom Longitude (generated) Marks card, click Tooltip.

An Edit Tooltip dialog box opens.

12. Copy the following text and paste it into the Edit Tooltip dialog box, and then click OK:

Total <State (group)> region sales: <SUM(Sales)>

The tooltip looks similar to this:



Congrats! You've created a dual axis map! There's only one thing left to do.

For more information about dual axis maps, see [Dual Axis Mapping - Many Ways](#) on the Tableau Community.

[Back to top](#)

Step 11: Customize how others can interact with your map

Now that you have created your map view, you can customize how people will interact with it. For example, you might now want anyone to be able to zoom in or out of your map, or pan. Or perhaps you want to display a map scale? You can customize these two options and more in the Map Options dialog box.

To customize how others can interact with your map:

1. Select Map > Map Options.
2. In the Map Options dialog box that appears, do the following:
 - Select Show Map Scale.
 - Clear Show Map Search.
 - Clear Show View Toolbar.

POSSIBLE QUESTIONS

PART-B

(5 X 6=30)

1. Describe about Graph databases Neo4J
2. Explain about Importing data into neo4j
3. Describe about Data visualization
4. Explain about Visualizations neo4j

PART-C

(1 X 10 =10)

1. Describe about Connecting your data
2. Explain about Creating Calculation
3. Explain about Creating Visual analytics with Tableau



KARPAGAM ACADEMY OF HIGHER EDUCATION

Department of Computer Science

I M.Sc(CS) (BATCH 2018-2020)

BIG DATA ANALYTICS

PART-A OBJECTIVE TYPE/ MULTIPLE CHOICE QUESTIONS

ONLINE EXAMINATIONS

ONE MARK QUESTIONS

S.No	UNIT-5	choice 1	choice 2	choice 3	choice 4			Answer
1	_____will initiate the mapper	Task tracker	Job tracker	Combiner	Reducer			Task tracker
2	Categorize the following to the following datatype	JSON files – Semi-structured	Word Docs , PDF Files , Text files – Unstructured	Email body – Unstructured	Data from enterprise systems (DB, CRM) – Structured			Email body – Unstructured
3	Hadoop is a framework that allows the distributed processing of _____	Small Data Sets	Semi-Large Data Sets	Large Data Sets	Large and Small Data sets			Large Data Sets
4	Sqoop ingest data from _____	Linux File Directory	Oracle &MySQL	HBase	MySQL			Oracle &MySQL
5	Identify the batch processing scenarios from following:	Sliding Window Averages Job	Facebook Comments Processing Job	Inventory Dynamic Pricing Job	Fraudulent Transaction Identification Job			Inventory Dynamic Pricing Job
6	_____ is true about Name Node	It is the Master Machine of the Cluster	It is Name Node that can store user data	Name Node is a storage heavy machine	Name Node can be replaced by any Data Node Machine			It is the Master Machine of the Cluster
7	_____are NOT metadata items	List of HDFS files	HDFS block locations	Replication factor of files	File Records distribution			File Records distribution

8	_____decides number of Mappers for a MapReduce job	File Location	mapred.map.tasks parameter	Input file size	Input Splits			Input file size
9	Name Node monitors block replication process of _____	TRUE	FALSE	replication	data node			FALSE
10	_____ true for Hadoop Pseudo Distributed Mode	It runs on multiple machines	Runs on multiple machines without any daemons	Runs on Single Machine with all daemons	Runs on Single Machine without all daemons			Runs on Single Machine with all daemons
11	Which of following statement(s) are correct?	Master and slaves files are optional in Hadoop 2.x	Master file has list of all name nodes	Core-site has hdfs and MapReduce related common properties	hdfs-site file is now deprecated in Hadoop 2.x			Core-site has hdfs and MapReduce related common properties
12	_____ is true for Hive	Hive is the database of Hadoop	Hive supports schema checking	Hive doesn't allow row level updates	Hive can replace an OLTP system			Hive doesn't allow row level updates
13	_____ is the highest level of Data Model in Hive	Table	View	Database	Partitions			Database
14	Hive queries response time is in order of _____	Hours at least	Minutes at least	Seconds at least	Milliseconds at least			Seconds at least
15	Managed tables in Hive _____	Can load the data only from HDFS	Can load the data only from local file system	Are useful for enterprise wide data	Are Managed by Hive for their data and metadata			Are Managed by Hive for their data and metadata

16	Partitioned tables in Hive _____	Are aimed to increase the performance of the queries	Modify the underlying HDFS structure	Are not useful if the filter columns for query are different from the partition columns	HDFS			Are aimed to increase the performance of the queries
17	Hive UDFs can only be written in Java	True	False					False
18	Hive can load the data from _____	Local File system	HDFS File system	Output of a Pig Job	File system			HDFS File system
19	HBase is a key/value store. Specifically it is _____	Sparse	Multi-dimensional	Distributed	Consistent			Multi-dimensional
20	_____ is the outer most part of HBase data model	Database	Table	Row key	Column family			Database
21	Which of the following is/are true:	HBase table has fixed number of Column families	HBase table has fixed number of Columns	HBase doesn't allow row level updates	HBase access HDFS data			HBase table has fixed number of Column families
22	Data can be loaded in HBase from Pig using _____	PigStorage	SqoopStorage	BinStorage	HbaseStorage			HbaseStorage
23	Sqoop can load the data in HBase _____	True	False					True
24	_____ APIs can be used for exploring HBase tables	HBaseDescriptor	HBaseAdmin	Configuration	HTable			HTable
25	_____ tables in HBase holds the region to key mapping	ROOT	.META.	MAP	REGIONS			.META.

26	_____ is the data type of version in HBase	INT	LONG	STRING	DATE			LONG
27	_____ is the data type of row key in HBase	INT	STRING	BYTE	BYTE[]			BYTE[]
28	HBase first reads the data from _____	Block Cache	Memstore	HFile	WAL			Memstore
29	The High availability of Namenode is achieved in HDFS2.x using _____	Polled Edit Logs	Synchronized Edit Logs	Shared Edit Logs	Edit Logs Replacement			Shared Edit Logs
30	The application master monitors all Map Reduce applications in the cluster _____	True	False					False
31	HDFS Federation is useful for the cluster size of _____	>500 nodes	>900 nodes	> 5000 nodes	> 3500 nodes			> 5000 nodes
32	Hive managed tables stores the data in _____	Local Linux path	Any HDFS path	HDFS warehouse path	None of the above			HDFS warehouse path
33	On dropping managed tables, Hive _____	Retains data, but deletes metadata	Retains metadata, but deletes data	Drops both, data and metadata	Retains both, data and metadata			Drops both, data and metadata
34	Managed tables don't allow loading data from other tables.	True	False					False
35	External tables can load the data from warehouse Hive directory.	True	False					True
36	On dropping external tables, Hive _____	Retains data, but deletes metadata	Retains metadata, but deletes data	Drops both, data and metadata	Retains both, data and metadata			Retains data, but deletes metadata

37	Partitioned tables can't load the data from normal (partitioned) tables	True	False					False
38	The partitioned columns in Hive tables are _____	Physically present and can be accessed	Physically absent but can be accessed	Physically present but can't be accessed	Physically absent and can't be accessed			Physically absent but can be accessed
39	Hive data models represent _____	Table in Metastore DB	Table in HDFS	Directories in HDFS	None of the above			Directories in HDFS
40	_____ is the earliest point at which the reduce method of a given Reducer can be called.	As soon as at least one mapper has finished processing its input split.	As soon as a mapper has emitted at least one record.	Not until all mappers have finished processing all records.	It depends on the InputFormat used for the job.			Not until all mappers have finished processing all records.
41	Apache Cassandra is a _____	SQL	NoSQL	NewSQL	Oracle			NoSQL
42	Cassandra uses a protocol of _____	gossip	inter-gossip	gossip	internal-gossip			gossip
43	A _____ determines the number of partitions	Client request	Snitch	Partitioner	non-partitioner			Snitch
44	User accounts may be managed by _____	Hive	Cassandra	Sqoop	Lucene			Cassandra
45	Authorization capabilities are managed by _____	COMMIT	GRANT	ROLLBACK	TRIGGER			GRANT
46	Client-to-node encryption is supported by _____	SSL	SSH	SSN	SLS			SSL
47	Using _____ for client-to-node encryption	qlshrc	cqlshrc	cqlshrc	qlc			cqlshrc
48	Internal authentication is supported by _____	system_auth	system_auth	system.credentials	sys_auth.credentials			system_auth.credentials
49	A _____ grants permissions to users	keyspace	superuser	sudouser	unuser			superuser
50	_____ is one of the authentication mechanisms in Cassandra	Cassandra	Cassandra	Cassandra	Authorizer			Cassandra Authorizer

51	Cassandra creates a ____	directory	subdirectory	domain	path			subdirectory
52	When _____, c	subtable	memtable	intable	memorytable			memtable
53	Data in the commit log	SSHables	SSTable	Memtables	SLSTable			SSTable
54	For each SSTable, Ca	memory	partition	in memory	synchronize			partition
55	Cassandra marks data	tombston	combstone	tenstone	stone			tombstone
56	Tombstones exist for a	gc_grace	gc_grace_	gc_grace_se	gc_grace_hours			gc_grace_seconds
57	_____ is a Cassar	Hinted ha	Hinted har	Tombstone	Hinted tomb			Hinted handoff
58	Cassandra searches th	partition	partition s	partition sea	partition file			partition summary
59	You configure sample	index_tin	index_inte	index_secs	indexed			index_interval
60	The compression offse	1-3	10-16	20-22	0-1			1-3
61	Neo4J is called	graph database	Hinted har	Hinted tomb	synchronize			graph database
62	In which language Neo4G is written	Java lang	tombstone	combstone	tenstone			Java langu
63	Which query language is used by Neo4J	Cypher Q	Crystal Que	Cypher Quit L	Cypher Query Length			Cypher Qu
64	The first version of Neo4j 1.0	Neo4j 1.0	Neo4j 2.0	Neo4j 3.0	Neo4j 4.0			Neo4j 1.0
65	it was released in	Feb, 2010	Feb, 2000.	Feb, 2012.	March, 2010.			Feb, 2010.
66	Neo4J is a very popu	Graph Da	subdirecto	domain	path			Graph Dat
67	What is CQL	Cypher Q	Crystal Que	Cypher Quit L	Cypher Query Length			Cypher Qu
68	how you can run CQ	“\$”	%	#	?	!		“\$”
69	types of object cache	two	three	four	five			two

Query Language (CQL)

Query Language