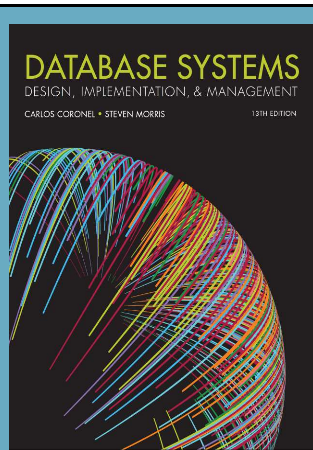
INFS5710 IT Infra. for BA



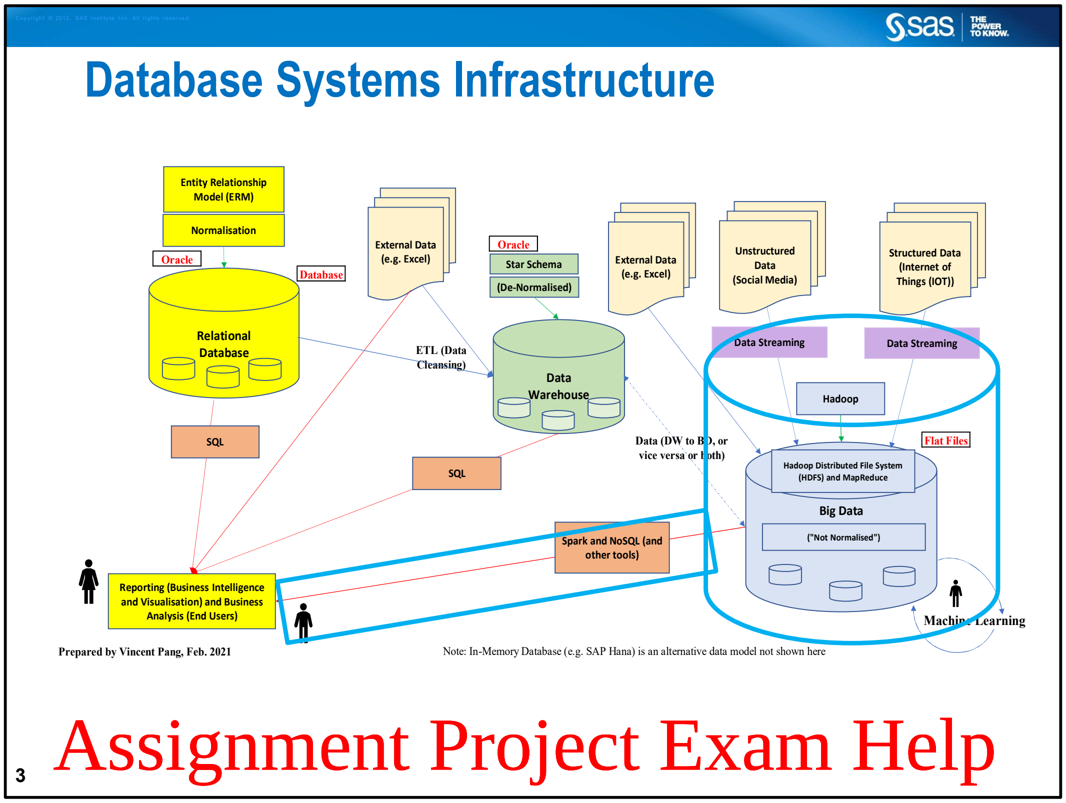
Chapter 14

Hadoop, MapReduce and NoSQL

14-2 to 14-3

pp. 664 - 679

Last week, we were looking at Big Data, this week we will be looking at Hadoop, MapReduce and NoSQL.



Today, we will cover the last bit you will learn in this course on Big Data. Now, you see how everything in the diagram is connected.

In summary, you have learned normalisation design of the database in Week 4 using ERD, i.e. Entity-Relationship Diagram, which you explored in Week 2 and 3. You learned to use Oracle to create an ERD in the lab.

Moreover, you have been learning SQL or Sequel in the Lab workshops.

You learned about data warehouse in Week 5.

Last week, you learned characteristics of Big Data and how Big Data has influenced in today's society in the video, "The Human Face of Big Data".

Again, there are extra materials for this lecture – all the text should be on slides.

This week, we will talk about Hadoop, MapReduce and NoSQL behind the building of big data. Also, on how data can be retrieved from relational database or saved the data back to the relational database. A data warehouse could be a relational database.

Hadoop

A software framework provides a standard way to build and deploy applications

- De facto standard for most Big Data storage and processing
- Java-based framework for distributing and processing very large data sets across clusters of computers
- Most important components:
  - **Hadoop Distributed File System (HDFS)**: Low-level distributed file processing system that can be used directly for data storage
  - **MapReduce:** Programming model that supports processing large data sets

Hadoop Distributed File System (HDFS)

- Approach based on several key assumptions:
  - *High volume* - Default block sizes is 64 MB and can be configured to even larger values
    Store big data in blocks stored in multiple devices

    Why? To avoid fragmentation Issues.

    scalability becomes important
  - *Write-once, read-many* - Model simplifies concurrency issues and improves data throughput

    automatic
  - *Streaming access* - Hadoop is optimized for batch processing of entire files as a continuous stream of data
    (from start to the end of each file w/o random seek)

    failure
  - *Fault tolerance* – HDFS is designed to replicate data across many different devices so that when one fails, data is still available from another device

5

Hadoop Distributed File System (HDFS)

- Uses several types of nodes (computers):
  - Data node store the actual file data
  - Name node contains file system metadata
  - Client node makes requests to the file system as needed to support user applications
  - store metadata
  - Data node communicates with name node by regularly sending **block reports** and **heartbeats**

what blocks are in data node

to inform name node the file status in the data node

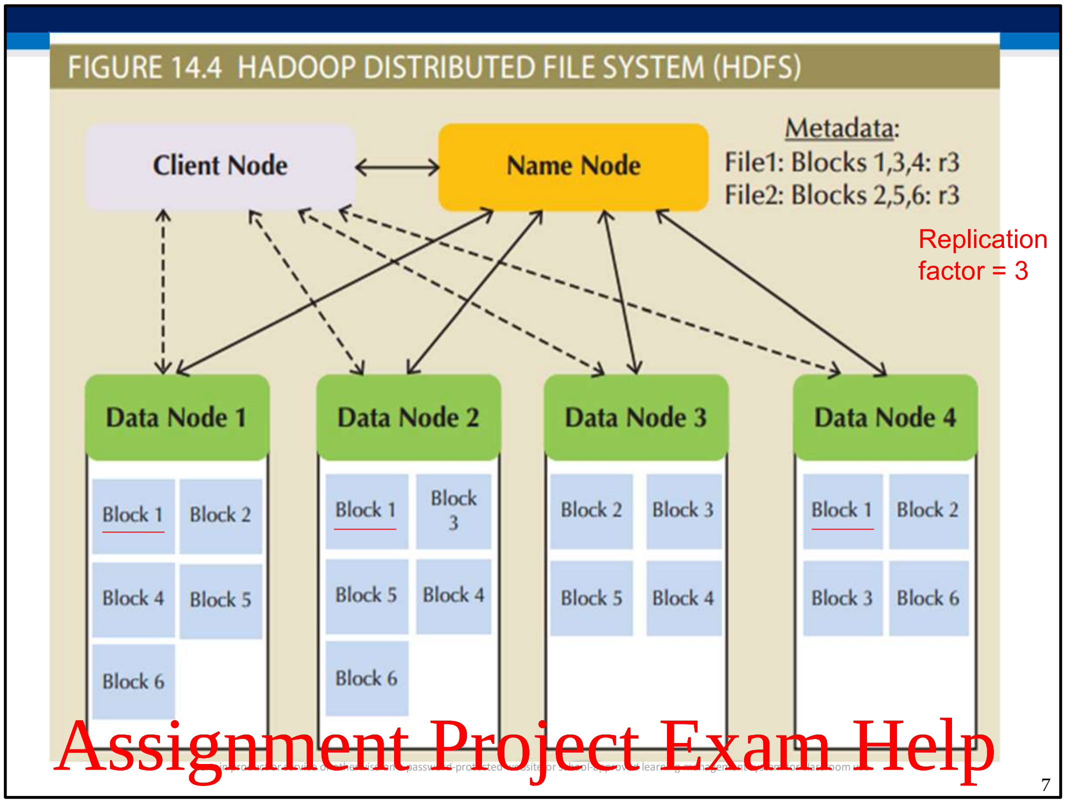


FIGURE 14.4 HADOOP DISTRIBUTED FILE SYSTEM (HDFS)
Client Node
Name Node
Metadata:
File1: Blocks 1,3,4: r3
File2: Blocks 2,5,6: r3
Replication factor = 3
Data Node 1
Block 1
Block 2
Block 4
Block 5
Block 6
Data Node 2
Block 1
Block 3
Block 5
Block 4
Block 6
Data Node 3
Block 2
Block 3
Block 5
Block 4
Data Node 4
Block 1
Block 2
Block 3
Block 6
7

MapReduce

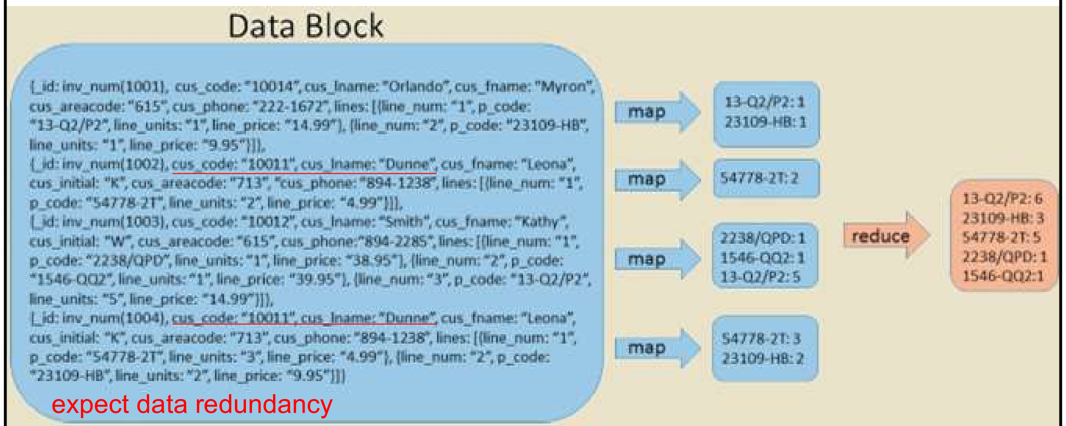
- Framework used to process large data sets across clusters   divide and conquer
  - Breaks down complex tasks into smaller subtasks, over nodes performing the subtasks and producing a final result
  - **Map** function takes a collection of data and sorts and filters it into a set of key-value pairs (key, value) or (attribute, value)
    - **Mapper** program performs the map function
  - **Reduce** summaries results of map function to produce a single result   Need an “objective”, e.g., sum, mean, max, or min.
    - **Reducer** program performs the reduce function

MapReduce

- Implementation complements HDFS structure
- Uses a **job tracker** or central control program to accept, distribute, monitor and report on jobs in a Hadoop environment divide and conquer
- **Task tracker** is a program in MapReduce responsible for reducing tasks on a node
- System uses **batch processing** (a sequence of tasks) which runs tasks from beginning to end with no user interaction

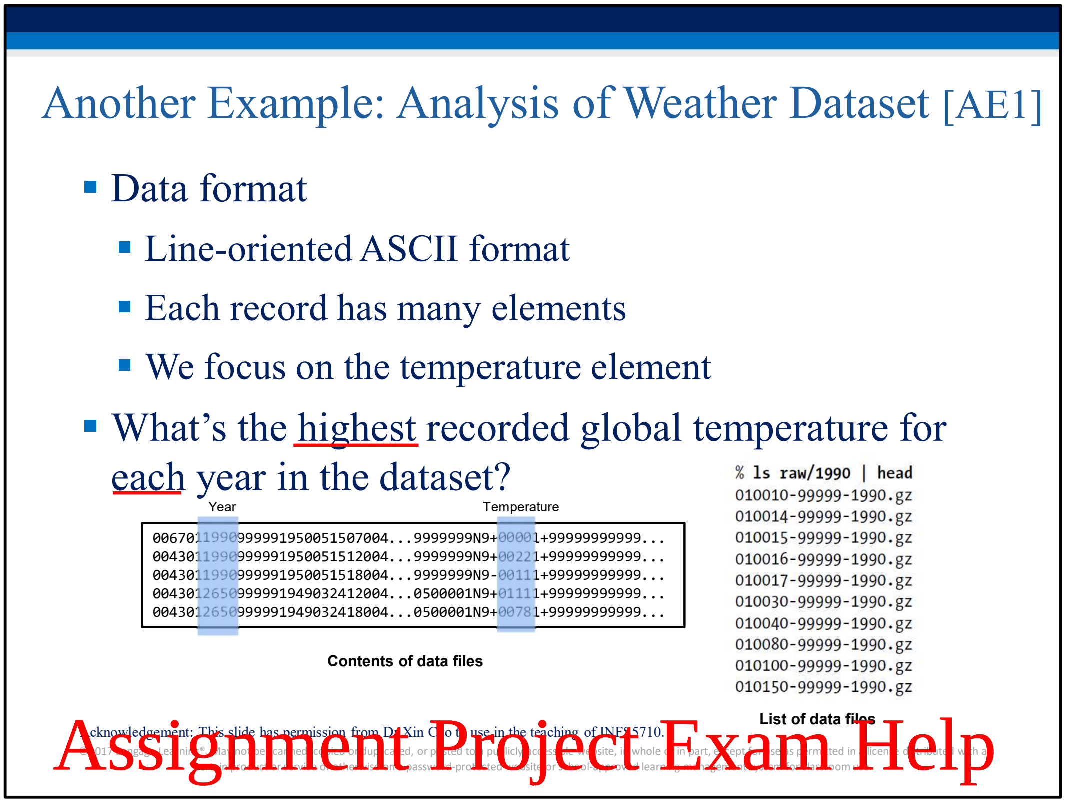
Determine the total number of units for each product that has been sold.

This task would be straightforward if the invoice data are stored in a relational DB.



key-value pairs

The task is to look for p_code and line_unit



This is example by my colleague (Xin). We co-teach in another course on Big Data.

If you want to gain more big data technical skills, you can enrol to COMP9313 (PG) in CSE.

After listening to his presentation, I think his example is better - explaining in more details how things work underneath.

This is a bit more details and "heavier" than the previous example. It has a bit newer materials and more up-to-date than the textbook.

In this example, we are looking maximum temperature for each year. In the input file, we can see year and temperature are different location on each row of the input file.

Solve this problem on one node [AE2]

- Keep a hash table <Year, Temperature>
- Read the data line by line
- For each line: get the year and temperature, check the current maximum temperature for the year, and update it accordingly

Acknowledgement: This slide has permission from Dr Xin Cao to use in the teaching of INFS5710.

We first look at the one node, i.e. one data node, or how the data is stored.

The process is start with a hash table. A hash table is a table that uses a hash function to compute an index in the table, so you can search very quickly.

Then, you read the data line by line. For each line, you retrieve the year and temperature.

Lastly, you compare the current maximum temperature for the year, and if it is higher than the current maximum, then you update the table accordingly.

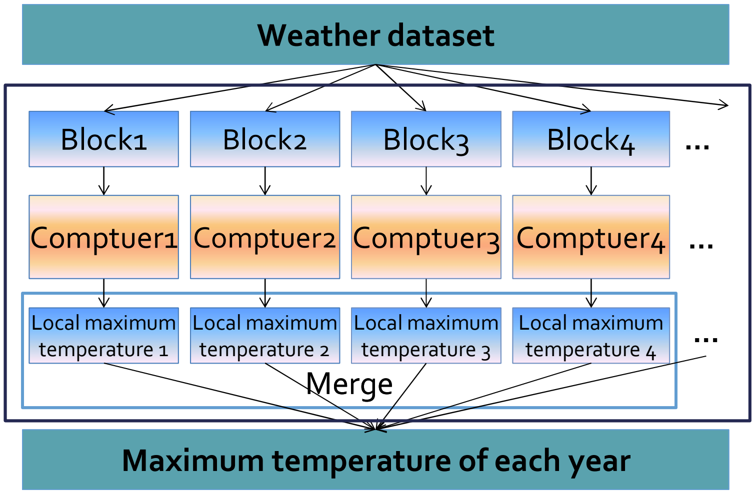
Solve this problem on multiple nodes [AE3]

- You need to first divide the data into several parts and distribute them to the nodes
- On each node, you need to maintain a hash table <Year, Temperature>
- The nodes do the following task in parallel: for each line, get the year and temperature, check the current maximum temperature for the year, and update it accordingly
- After all the nodes find the "local" maximum temperature store in hash tables, aggregate the results on one node to compute the maximum temperature of each year

Acknowledgement: This slide has permission from Dr Jin Cao to use in the teaching of INFS5710.

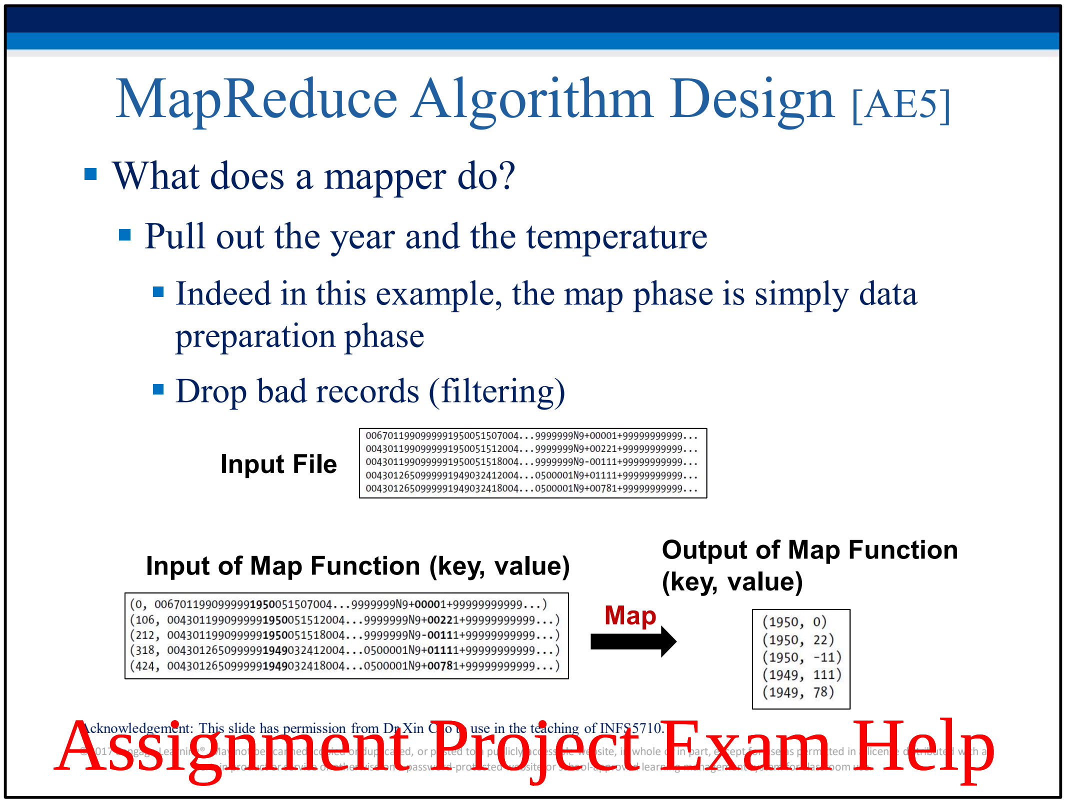
Same as before, except you do on multiple nodes.

Maximum Temperature [AE4]



Acknowledgement: This slide has permission from Dr Xin Cao to use in the teaching of INFS5710.

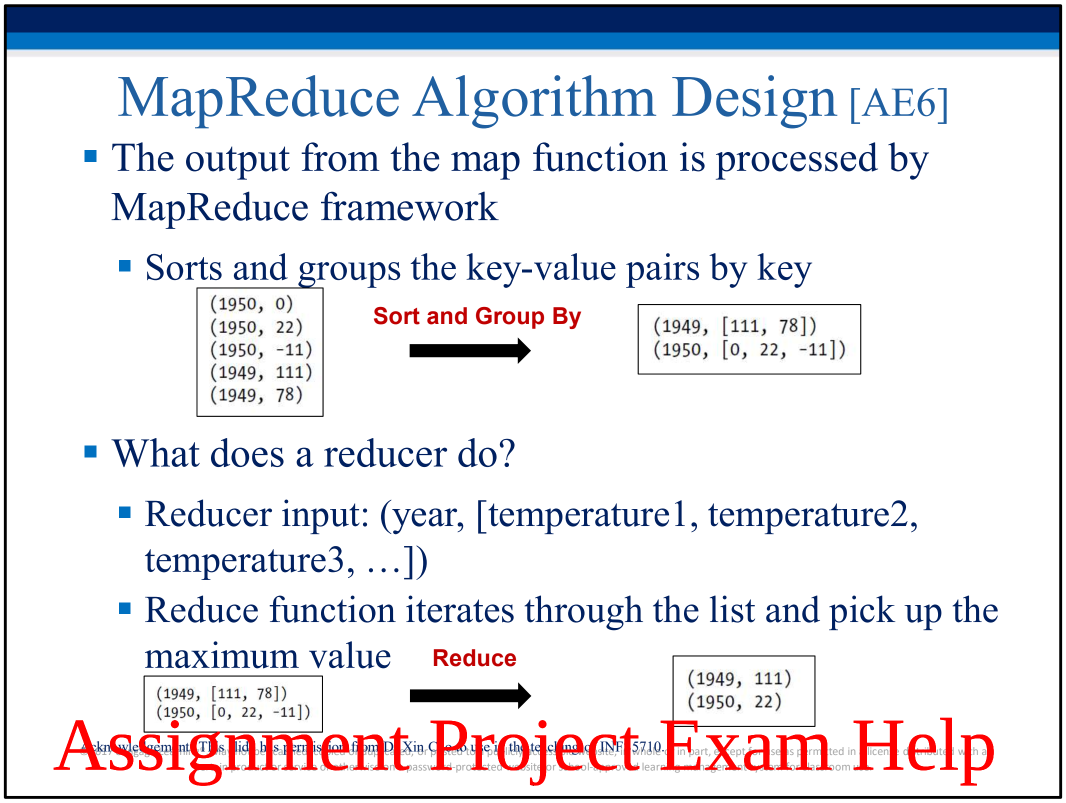
This is an overall picture. All the nodes read the lines and then update the maximum temperature of each year.



Now, in more details on how this is done.

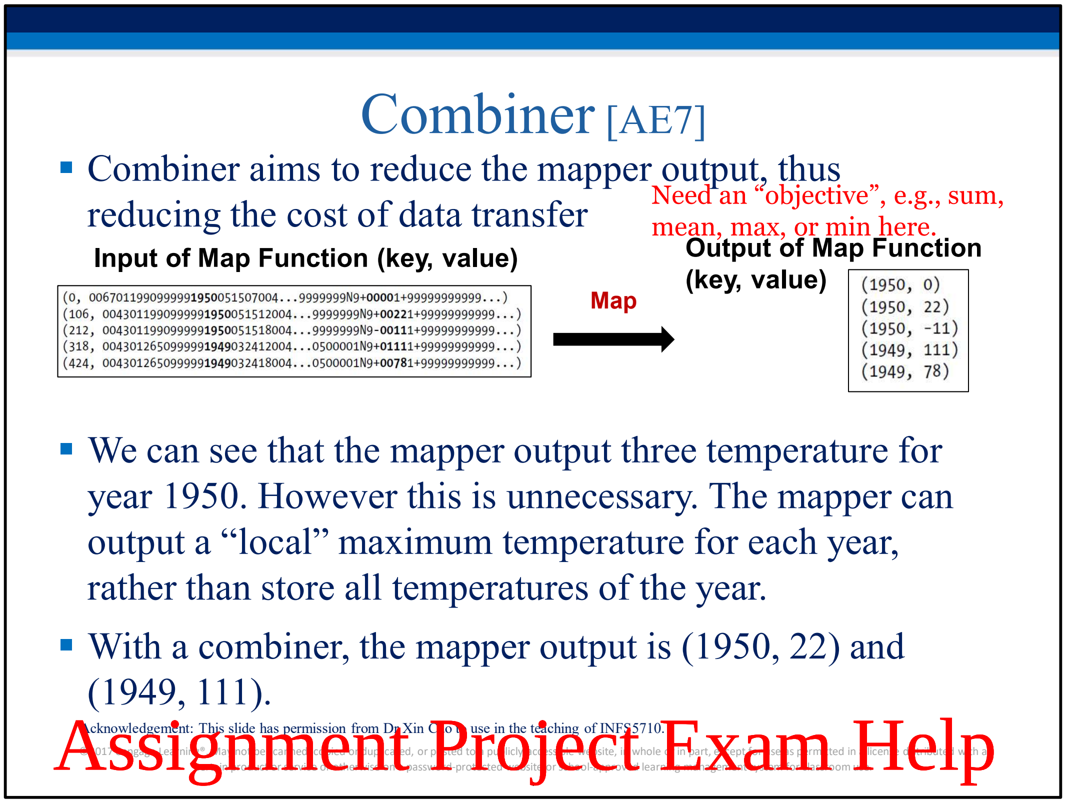
After reading the lines from the input file, the map function will map the year and temperature as output of the Map Function (key, value), i.e. (year, temperature).

In the example, Map function manages to retrieve the data into year and temperature as (1950,0), (1950,22). (1950,-11) etc.



The data is then sorted and grouped into key-value pairs by key, so, now you have (1950, **[0, 22, -11]**). The values in the square bracket only appear once. If you have the same value, such as 22, it only keeps one.

The next stage is to use the reduce function. The reducer will then reduce the maximum temperature for each year to (1950, 22).



This is not in the text book

Let's look into more details on how this is done. This Combiner step comes after Map phase but before Reduce Phase.

The Combiner sorts and groups the year together, i.e. all the keys with 1950s are grouped together.

In this example, the mapper output will be (1950, 22) and (1949, 111). You will have maximum temperature for all the years from the original input file.

Note, here we are only looking at one node only. Other nodes will do exactly like this one, it might come out different output as it might different blocks. Data Node 2, say, has (1950, 10), and Data Node 2 might have (1959, 5).

Partitioner [AE8]

- Partitioner controls the partitioning of the keys of the intermediate map outputs.
  - The key (or a subset of the key) is used to derive the partition, typically by a hash function.
  - The total number of partitions is the same as the number of reduce tasks for the job.
    - This controls which reduce tasks an intermediate key (and hence the record) is sent to for reduction.
- System uses HashPartitioner by default:
  - hash(key) mod R, where R is the number of partitions

    The remainder of hash(key) divided by R, which can be 0, 1, 2, ..., R-1.

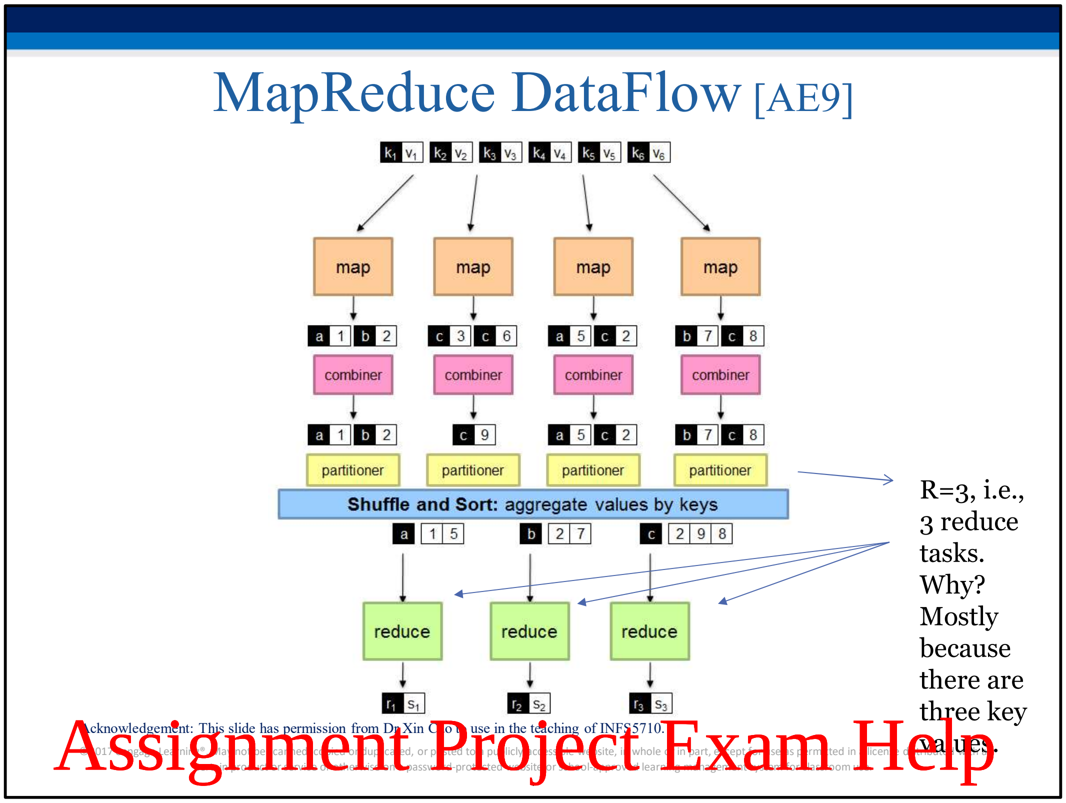
Again, this is not in the text book, partitioner controls the partitioning of the keys of the intermediate map outputs. This partition phase, again, takes place after the Map phase but before the Reduce phase.

In this step, you look at all the nodes, and find all the temperature for the same year, i.e. you look at year 1950 in Data Node 1, Data Node 2 and so on. You will find the highest temperature from these nodes, and put into one partition.

The number of partitioners is equal to the number of reducers.

What we are saying here is a partitioner will divide the data according to the number of reducers.

Therefore, the data passed from a single partitioner is processed by a single Reducer.

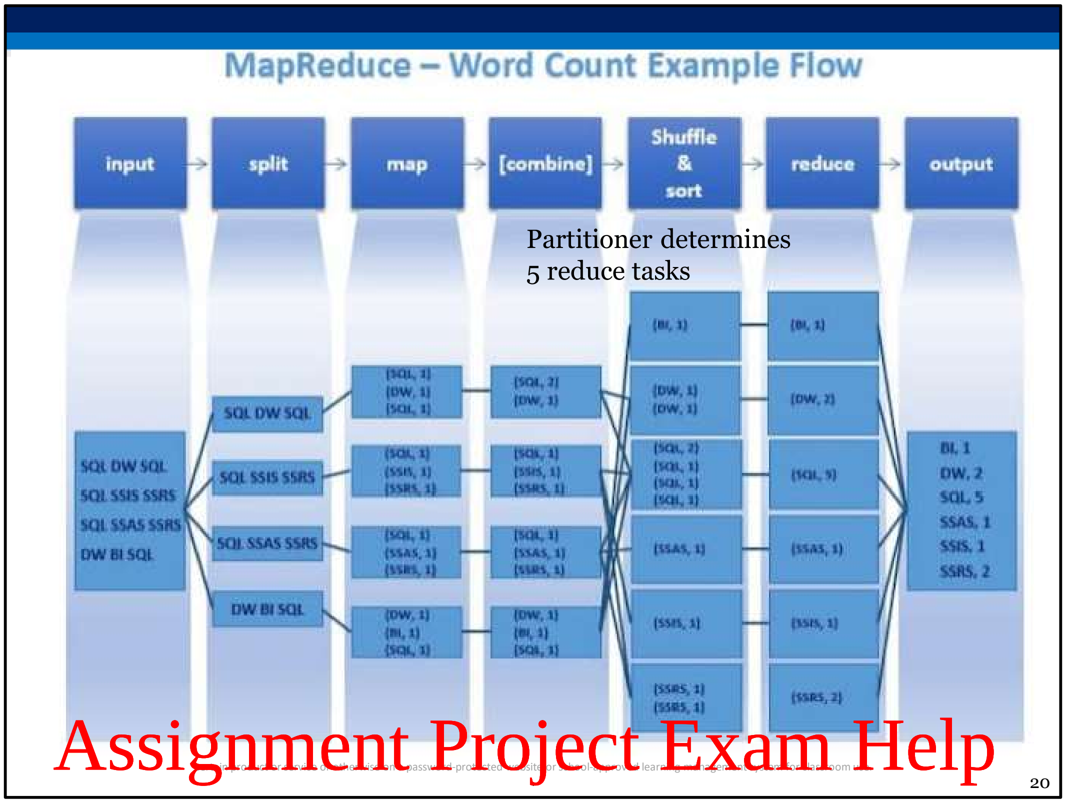


This diagram shows the data nodes are run in parallel and generate different results for a, b, and c.

After mapping process, you can see you have a=1 and b=2; c=3 and c=6; a=5 and c=2; b=7 and c=6.

After combiner process, you can see you have c=3 and c=6 become c=9.

After partitioner process, you can see you have a has 1 and 5; b has 2 and 7, and c has 2, 9 and 8.


MapReduce – Word Count Example Flow
input
split
map
[combine]
Shuffle & sort
reduce
output
Partitioner determines
5 reduce tasks
SQL DW SQL
SQL SSIS SSRS
SQL SSAS SSRS
DW BI SQL
BI, 1
DW, 2
SQL, 5
SSAS, 1
SSIS, 1
SSRS, 2
20

Hadoop Ecosystem (HE1)

- Hadoop is a **low-level tool**, which requires technical skills, considerable effort to create, manage, and use, it presents quite a few obstacles.
- Most organisations that use Hadoop as a set of other related products that interact and complement each other to produce an entire ecosystem of applications and tools **on top of Hadoop**.
  - These applications and tools **will help less technical users** who do not have technical skills to do low-level tool.
- Like any ecosystem, the interconnected pieces are constantly evolving and their relationships are changing, so it is a rather fluid situation

As stated on the slide...

Hadoop Ecosystem (HE2)

- MapReduce **Simplification Applications**:
  - *Hive* is a data warehousing system that sites on top of HDFS and supports its own SQL-like language
  - *Pig* compiles a high-level scripting language (Pig Latin) into MapReduce jobs for executing in Hadoop
- Data Ingestion Applications:
  - *Flume* is a component for ingesting data in Hadoop collecting

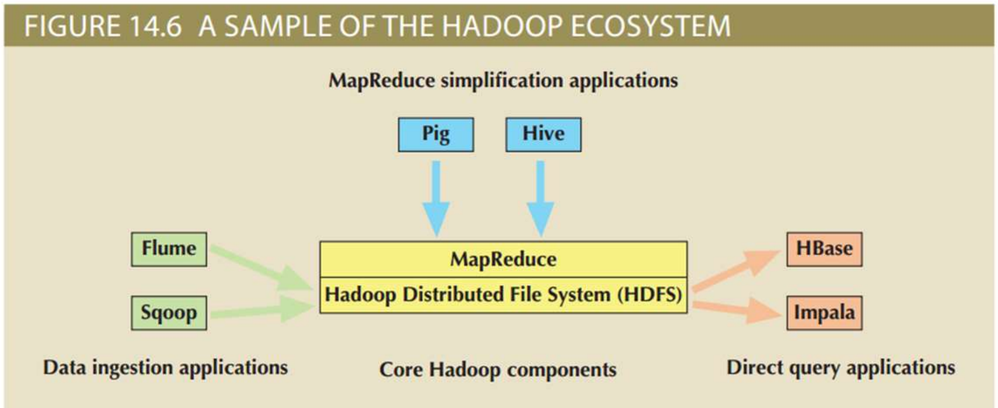
    Hive vs. Pig vs. MapReduce
  - *Sqoop* is a tool for converting data back and forth between a **relational database** and the HDFS
- Direct query applications
  - *HBase*: column-oriented NoSQL database designed to sit on top of the HDFS that quickly processes sparse datasets
  - *Impala*: the first SQL on Hadoop application

22

As stated on the slide…

Let's see next slide.

Hadoop Ecosystem (HE3)



You have learned the core Hadoop Components, namely MapReduce and HDFS, from the previous slides – you have to program in Java or Python.

MapReduce **<u>Simplification Applications</u>**:

Hive and ***Pig*** are designed to be more users friendly and require less technical skills and less time required to achieve the output.

Hive is a data warehousing system that sites on top of HDFS. It is not a relational database but supports its own SQL-like language. Hive SQL is like SQL can submit Hive Query to communicate with MapReduce. It is good for a large dataset but less efficient when it is after a small dataset and responses quickly.

Pig compiles a high-level scripting language called Pig Latin. It is written in scripting language, like Hive, it communicates with MapReduce jobs to execute in Hadoop. Pig is useful for query processing. The procedural language is required the user to specify how to data is to be manipulated. This is very useful for performing data transformation or ETL.

For example, one test 10 lines of Pig Latin is similar in testing 200 lines of Java. It takes about 15 minutes in writing a Pig Latin script file might take 4 hours to write in Java.

<u>Data Ingestion Applications</u>:

One of the issues is getting data from the existing systems into the Hadoop cluster. The applications have been developed to "ingest" or gather this data into Hadoop.

> ***Flume*** is a component for ingesting data in Hadoop. It is designed primarily for how much things harvesting large sets of data from server log files, like clickstream data from web server logs.
>
> ***Sqoop*** is a tool for converting data back and forth between a **<u>relational database</u>** and the HDFS.
>
> While ***Flume*** works primarily with log files, ***Sqoop*** works with relational database such as Oracle. ***Flume*** operates in one direction only, whereas ***Sqoop*** works on both directions of data transfer. That is the **<u>blue rectangles</u>**.

<u>Direct query applications</u>:

Provide faster query access than is possible through MapReduce. These applications interact with HDFS directly, instead of going through the MapReduce processing layer.

> ***HBase***: column-oriented NoSQL database designed to sit on top of the HDFS that quickly processes datasets. It does not support SQL or SQL-like languages. The system does not reply on MapReduce jobs, so it avoids the delays caused by batch processing, so it can process a small dataset or smaller subsets of the data.
>
> ***Impala***: the first SQL on Hadoop application (by Cloudera). With Impala, you can write a SQL queries directly against the data while it is still in HDFS. Impala makes heavy use of in-memory caching on data nodes.

“Not Only SQL”
It means it may
support SQL.

NoSQL (1 of 7)

non-SQL
non-relational

it is some kind of database

- Name given to non-relational database technologies developed to address Big Data challenges

4 categories of NoSQL products

TABLE 14.3

NoSQL DATABASES

NoSQL CATEGORY	EXAMPLE DATABASES	DEVELOPER
Key-value database	Dynamo Riak Redis Voldemort	Amazon Basho Redis Labs LinkedIn
Document databases	MongoDB CouchDB OrientDB RavenDB	MongoDB, Inc. Apache OrientDB Ltd. Hibernating Rhinos
Column-oriented databases	HBase Cassandra Hypertable	Apache Apache (originally Facebook) Hypertable, Inc.
Graph databases	Neo4J ArangoDB GraphBase	Neo4j ArangoDB, LLC FactNexus

24

As stated on the slide...

NoSQL (2 of 7)

1. **Key-value (KV) databases** store data as a collection of key-value pairs organized as **buckets** which are the equivalent of tables

FIGURE 14.7 KEY-VALUE DATABASE STORAGE

a key is an identifier

Bucket = Customer

Key	Value
10010	"LName Ramas FName Alfred Initial A Areacode 615 Phone 844-2573 Balance 0"
10011	"LName Dunne FName Leona Initial K Areacode 713 Phone 894-1238 Balance 0"
10014	"LName Orlando FName Myron Areacode 615 Phone 222-1672 Balance 0"

a bucket

A key is a key but not a primary key. It does not have to be unique.

Though the bucket appears in tabular form in this figure, actually key-value pairs are not stored in a table-like structure.

NoSQL (3 of 7)

2. **Document databases** store data in key-value pairs in which the value components are tag-encoded documents grouped into logical groups called **collections**

FIGURE 14.8 DOCUMENT DATABASE TAGGED FORMAT

Collection = Customer

tag

Key	Document
10010	{LName: "Ramas", FName: "Alfred", Initial: "A", Areacode: "615", Phone: "844-2573", Balance: "0"}
10011	{LName: "Dunne", FName: "Leona", Initial: "K", Areacode: "713", Phone: "894-1238", Balance: "0"}
10014	{LName: "Orlando", FName: "Myron", Areacode: "615", Phone: "222-1672", Balance: "0"}

a collection

26

As seen in previous example, the data are in tag.

Document databases are "schema less". That is, they do not impose a predefined structure on the data stored.

Document databases are similar to Key-Value (KV) database and sometimes it can be said to be a subtype of KV database.

The key difference is how the data is stored. In KV database, the value can be lumped into a bucket, but in Document database, the values are tagged.

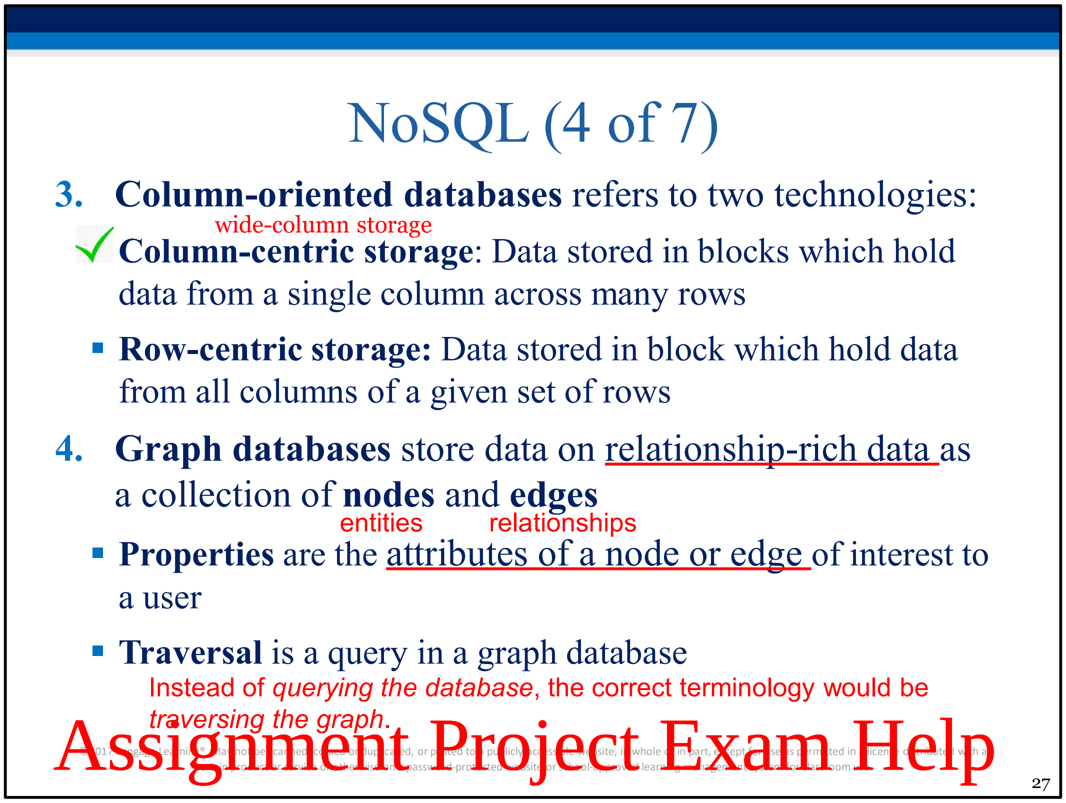
For example,

in KV database, a bucket can be "Lname Ramas Fname Alfred Initial A …." whereas

in document database, a collection can be {Lname: "Ramas", Fname: "Alfred", Initial: "A" …."

Examples include XML, JSON (JavaScript Object Notation)

Although all documents have tags, not all documents are required to have the same tags, so each document can have its own structure.



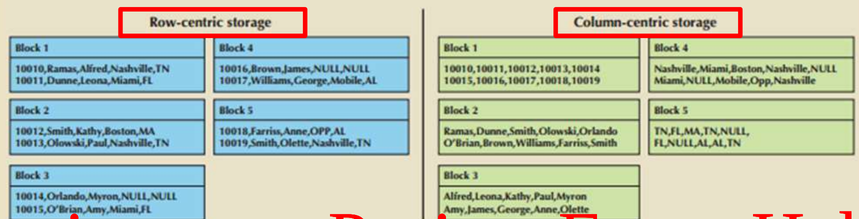
As stated on the slide...

NoSQL (5 of 7)

FIGURE 14.9 COMPARISON OF ROW-CENTRIC AND COLUMN-CENTRIC STORAGE

CUSTOMER relational table

Cus_Code	Cus_LName	Cus_FName	Cus_City	Cus_State
10010	Ramas	Alfred	Nashville	TN
10011	Dunne	Leona	Miami	FL
10012	Smith	Kathy	Boston	MA
10013	Olowski	Paul	Nashville	TN
10014	Orlando	Myron		
10015	O'Brian	Amy	Miami	FL
10016	Brown	James		
10017	Williams	George	Mobile	AL
10018	Farriss	Anne	Opp	AL
10019	Smith	Olette	Nashville	TN



28

In **traditional normalized tables**, only very relevant attributes are placed in the same table, it is more effective to manipulate or retrieve data in row-oriented storage. Cust_code is the primary key, and the rests are attributes.

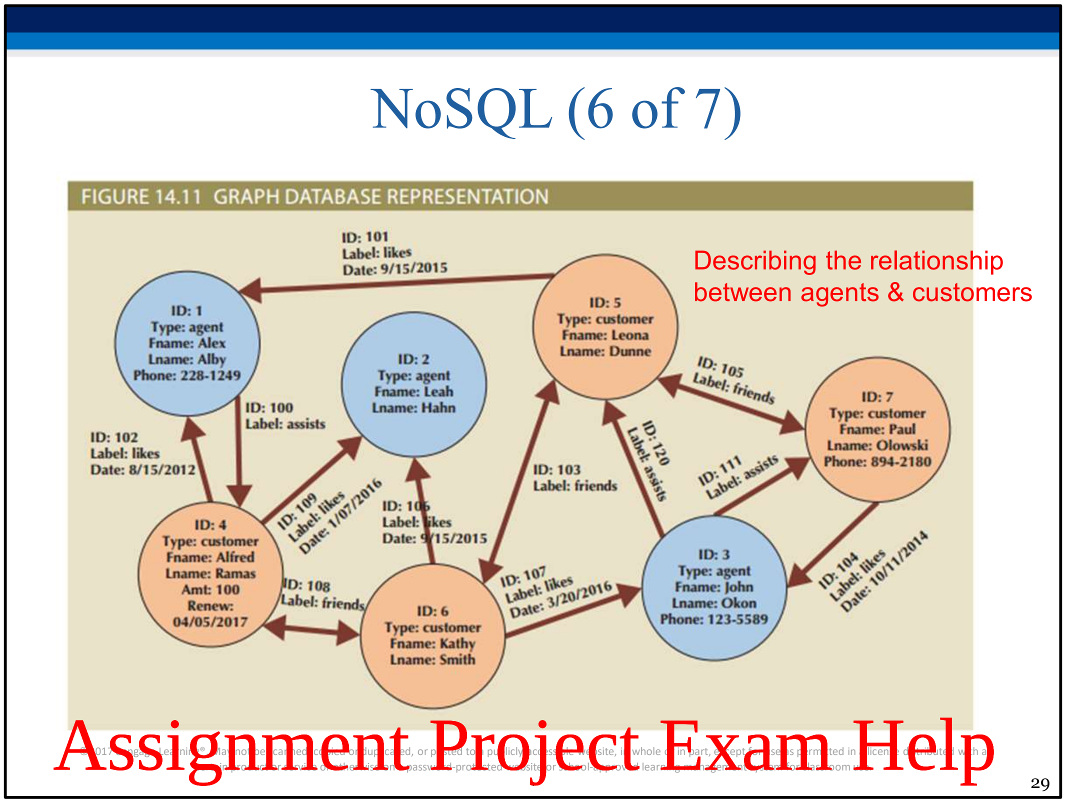
In the **Row-centric storage**, you out the whole row into a block.

In big data environment, the number of rows is much greater than the number of columns. Mostly, people retrieve a small set of columns across a large set of rows. Column-oriented database is more effective.

On the other hand, for the **Column-centric storage**, the columns are broken down and placed in different blocks. Please note that although they are placed in different columns, a row is still linked together across the blocks. The book says, "a row is spread across the blocks." That is, you start with 10010 in Block 1, you can get to Block 2, to get Ramas and so on. There is a reason why in column-centric, it is faster to get the data. For example, you want to find all the people who live in Florida or FL, you can go to Block 5 to find all the "FL" and then go to Block 2 and Block 3 to get the names. In this example, there are two people live in Florida, namely Leona Dunne and Amy O'Brien.

Another example, if we have a Block 6, say sales, we can just look at this column to sum or average the sales without having to look at other columns. Thus, this will speed up of getting the output.

You will ask, "isn't it the same as in a ***relational database***?" In theory, they look the same, but how the rows or columns are retrieved and tested for values are different. The data management system runs differently.



Graph databases store data on relationship-rich data as a collection of **nodes** and **edges.** This is based on Graph Theory, which is a mathematical and computer science field that models relationship or ***edges***, between objects called ***nodes***.

This is originated in the area of **social networks** such as in Facebook.

On a side issue, you probably see how Coronavirus is spread around the globe. This can be done using a complex Network theory developed by Watts–Strogatz based on "six degrees of separation" based on six degrees of Kevin Bacon concept. If you know someone, and someone knows another person, and then someone knows Kevin Bacon. It works out that there are the most six steps or six people to reach Kevin Bacon. One of the ladies who is not famous has lots of connections with stars and other people, they classified her the hub because she has lots of connections.

You saw the prediction of how virus is spread on TV. One of the network models used is probably based on Watts–Strogatz model, or similar network models. They built a Coronavirus network. For example, if a person with Coronavirus came from the cruise ship, then you probably will want to find out who the person s/he has contacted, you put the names the person associated with, who is sick and who is not, in the system. Thus, you will start building up a graph network. This is a side issue, which you might find interesting.

Properties are the attributes of a node or edge, e.g. Like, interest to a user

Traversal is a query in a graph database

- Graph databases do not scale out very well to clusters.
- The other 3 NoSQL DB models achieve clustering efficiency by making each piece of data relatively independent.
- Separating data into independent pieces across nodes in the cluster, often called sharding (partitioning), is what allows NoSQL to scale out effectively.

NoSQL (7 of 7)

- ***Aggregate awareness***: data is collected or aggregated around a central topic or entity
  - Examples include KV, document, and column family databases
  - Aggregate aware database models achieve clustering efficiency by making each piece of data relatively independent
- Graph databases, like relational databases, are ***aggregate ignorant***
  - Do not organize the data into collections based on a central entity

Aggregate aware means that the data is collected or aggregated around a central topic or entity. That is, it is arranged to how the data will be used.

Aggregate ignorant does not organise the data around a central entity based on how the data will be used.