DSECLZG522: Big Data Systems
Session 4 – BDA Lifecycle, CAP Theorem, NoSQL

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

Topics for today

- **Big Data Analytics lifecycle**
- Consistency, Availability, Partition tolerance
- CAP theorem
- NoSQL Introduction
- Classification of NoSQL databases

Big Data Analytics Lifecycle

- Big Data Analysis differs from traditional data analysis primarily
  - ✓ due to the volume, velocity and variety characteristics of the data being processed
- A step-by-step methodology is needed to organize the activities / tasks involved with
  - ✓ Acquiring
  - ✓ Storing
  - ✓ Processing
  - ✓ Analyzing
  - ✓ Repurposing data
- Explore a specific data analytics lifecycle that organizes and manages the tasks and activities associated with the analysis of Big Data

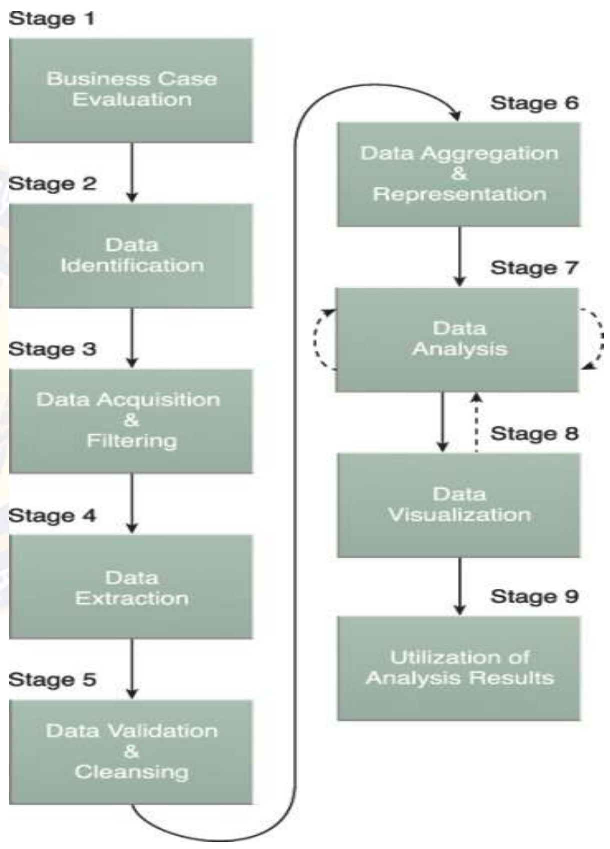
Example

1. A Market Research (MR) firm creates and runs surveys to understand specific market segments for their client, e.g. consumer electronics - LED TVs
2. These surveys contain questions that have structured : numeric, boolean, categorical, grade as well as unstructured : free form text answers
3. A survey is rolled out to selected users with various demographic attributes. The list of survey users could be provided by the client and/or MR firm
4. The results are collected and analyzed for business insights often using multiple tools and analysis techniques
5. The insights are curated and shared to create a presentation for the client of the MR firm
6. The client makes critical business decisions about their product based on the survey results

Big Data Analytics Lifecycle

Stages

1. Business Case Evaluation
2. Data Identification
3. Data Acquisition & Filtering
4. Data Extraction
5. Data Validation & Cleansing
6. Data Aggregation & Representation
7. Data Analysis
8. Data Visualization
9. Utilization of Analysis Results



1. Business Case Evaluation

- Based on business requirements, determine whether the business problems being addressed is really a Big Data problem
  - ✓ A business problem needs to be directly related to one or more of the Big Data characteristics of Volume, Velocity, or Variety.

High volume
Unstructured data

- Must begin with a **well-defined business case** that presents a clear understanding of
  - ✓ justification
  - ✓ motivation
  - ✓ goals of carrying out the analysis.

Find market fit
for new product

- A business case should be created, assessed and approved prior to proceeding with the actual hands-on analysis tasks.

What are the business questions ?
Define thresholds on survey stats

- Helps decision-makers to
  - ✓ Understand the business resources that will need to be utilized
  - ✓ Identify which business challenges the analysis will tackle.
  - ✓ Identify KPIs can help determine assessment criteria and guidance for the evaluation of the analytic results

2. Data Identification

- Main objective is to identify the datasets required for the analysis project and their sources
  - ✓ Wider variety of data sources may increase the probability of finding hidden patterns and correlations.
    - ✓ Caution: Too much data variety can also confuse - overfitting problem.
  - ✓ The required datasets and their sources can be internal and/or external to the enterprise.

- For internal datasets
  - ✓ A list of available datasets from internal sources, such as data marts and operational systems, are typically compiled and verified for data requirement

Identify respondents
Demographics
What questions to ask
Do we need other surveys

- For external datasets
  - ✓ A list of possible third-party data providers, such as data markets and publicly available datasets needs to be compiled
  - ✓ Data may be embedded within blogs or other types of content-based web sites, automated tools needs to be used to extract it – Web scraping tools

3. Data Acquisition & Filtering

- The data is gathered from all the data sources that were identified during the last stage
- The acquired data is then looked upon for
  - ✓ filtering / removal of corrupt data
  - ✓ removal of unusable data for analysis

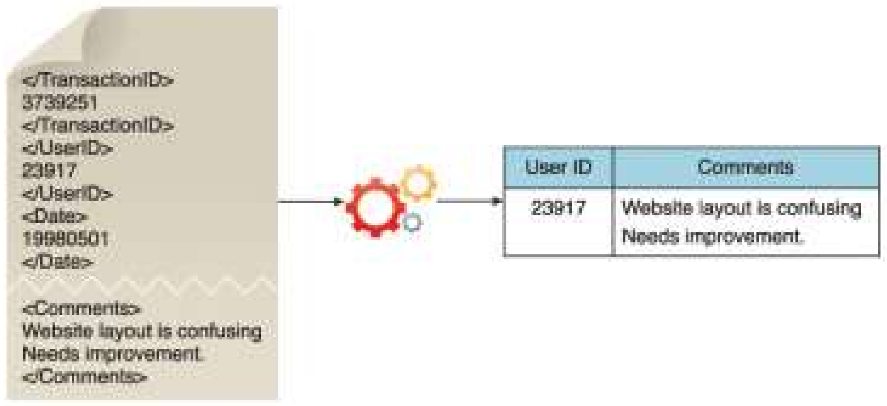
Clean bad data, e.g. empty responses
Junk text inputs
Filter a subset if we don't need to look at all attributes, all demographics

- In many cases involving unstructured external data, some or most of the acquired data may be irrelevant (noise) and can be discarded as part of the filtering process.
- "Corrupt" data can include records with missing or nonsensical values or invalid data types
  - ✓ Advisable to store a verbatim copy of the original dataset before proceeding with the filtering
- Data needs to be persisted once it gets generated or enters the enterprise boundary
  - ✓ For **batch analytics,** this data is persisted to disk prior to analysis
  - ✓ For **real-time analytics,** the data is analyzed first and then persisted to disk

4. Data Extraction

- Dedicated to extracting data and transforming it into a format that the underlying Big Data solution can use for the purpose of the data analysis
- The extent of extraction and transformation required depends on the types of analytics and capabilities of the Big Data solution.

Structure the unstructured responses



User ID	Comments
23917	Website layout is confusing Needs improvement.

Comments and user IDs are extracted from an XML document.

5. Data Validation & Cleansing

Validate survey responses
Contradictory answers
Identify population skews, e.g. responses have inherent gender bias so no point in making a gender based analysis
Codify certain columns for easier analysis

- Invalid data can skew and falsify analysis results

- Data input into Big Data analyses can be unstructured without any indication of validity
  - ✓ Complexity can further make it difficult to arrive at a set of suitable validation constraints
  - ✓ Dedicated stage is required to establish complex validation rules and removing any known invalid data.

- Big Data solutions often receive redundant data across different datasets.
  - ✓ This can be exploited to explore interconnected datasets in order to
    - assemble validation parameters
    - fill in missing valid data

- For batch analytics, data validation and cleansing can be achieved via an offline ETL operation
- For real-time analytics, a more complex in-memory system is required to validate and cleanse the data as it arrives from the source

6. Data Aggregation & Representation

- Dedicated to integrating multiple datasets together to arrive at a unified view
  - ✓ Need to merge data spread across multiple datasets through a common field
  - ✓ Needs reconciliation of data coming from different sources
  - ✓ Needs to identify the dataset representing the correct values.

Final joined data set (e.g. current with old survey or 3rd party demographics data) with certain aggregations done for downstream analysis

- Can be complicated because of :
  - ✓ Data Structure – Although the data format may be the same, the data model may be different
  - ✓ Semantics – A value that is labeled differently in two different datasets may mean the same thing, for example "surname" and "last name."
- The large volumes makes data aggregation a time and effort-intensive operation
  - ✓ Reconciling these differences can require complex logic that is executed automatically without the need for human intervention
- Future data analysis requirements need to be considered during this stage to help foster data reusability.

7. Data Analysis

- Dedicated to carrying out the actual analysis task, which typically involves one or more types of analytics
  - ✓ Can be iterative in nature, especially if the data analysis is exploratory
  - ✓ Analysis is repeated until the appropriate pattern or correlation is uncovered

- Depending on the type of analytic result required
  - ✓ Can be as simple as querying a dataset to compute an aggregation for comparison
  - ✓ Can be as challenging as combining data mining and complex statistical analysis techniques to discover patterns and anomalies or to generate a statistical or mathematical model to depict relationships between variables.

Various types of descriptive / predictive analysis on survey data to understand market fit for new product. Writing SQL on data and create charts. Build models on the data for hypothesis testing, prediction.

7. Confirmatory / Exploratory data analysis

- Confirmatory data analysis
  - ✓ A deductive approach where the cause of the phenomenon being investigated is proposed beforehand - a hypothesis
  - ✓ Data is then analyzed to prove or disprove the hypothesis and provide definitive answers to specific questions
  - ✓ Data sampling techniques are typically used
  - ✓ Unexpected findings or anomalies are usually ignored since a predetermined cause was assumed

- Exploratory data analysis
  - ✓ Inductive approach that is closely associated with data mining
  - ✓ No hypothesis or predetermined assumptions are generated
  - ✓ Data is explored through analysis to develop an understanding of the cause of the phenomenon
  - ✓ May not provide definitive answers
  - ✓ Provides a general direction that can facilitate the discovery of patterns or anomalies

Confirm findings or exception cases in the survey data through adhoc exploration. E.g. in how many cases young males like feature X but don't like feature Y.

8. Data Visualization

Visual results will need to be shared with the stakeholders for the new product launch. e.g. show top features that appeal to each segment of target product user (gender, age group).

- Dedicated to using data visualization techniques and tools to graphically communicate the analysis results for effective interpretation by business users
  - ✓ The ability to analyze massive amounts of data and find useful insights carries little value if the only ones that can interpret the results are the analysts.
  - ✓ Business users need to be able to understand the results in order to obtain value from the analysis and subsequently have the ability to provide feedback
- Provide users with the ability to perform visual analysis, allowing for the discovery of answers to questions that users have not yet even formulated
  - ✓ A method of drilling down to comparatively simple statistics is crucial, in order for users to understand how the rolled up or aggregated results were generated
- Important to use the most suitable visualization technique by keeping the business domain in context
  - ✓ Interpretation of result can vary based on the visualization shown

9. Utilization of Analysis Results

- Input for Enterprise Systems
  - ✓ Results may be automatically or manually fed directly into enterprise systems to enhance and optimize their behaviors and performance
  - ✓ Online store can be fed processed customer-related analysis results that may impact how it generates product recommendations

- Business Process Optimization
  - ✓ The identified patterns, correlations and anomalies discovered during the data analysis are used to refine business processes
  - ✓ Consolidating transportation routes as part of a supply chain process

- Alerts
  - ✓ Results can be used as input for existing alerts or may form the basis of new alerts
  - ✓ Alerts may be created to inform users via email or SMS text about an event that requires them to take corrective action

Topics for today

- Big Data Analytics lifecycle
- **Consistency, Availability, Partition tolerance**
- CAP theorem
- NoSQL Introduction
- Classification of NoSQL databases

Consistency

- Causes of consistency problem Big Data Systems
  - Big Data Systems use distributed (scale-out) computing
  - Big Data systems write replicas of a shard / partition
  - Any write needs to be updated on all replicas and it takes time
  - A read can happen in between from one or more replicas
  - The *inconsistency window* depends on communication delays between replicas and its sources
- Consistency scenarios
  - Do you allow a read of any replica in any node to always read the latest value written from any node ?
    - RDBMS / OLTP systems / Systems of Record
    - ACID (Atomicity, Consistency, Isolation, Durability)
  - Do you allow reads to return any value and eventually show the latest stable value
    - Some Big Data systems / Systems of Engagement e.g. social network comments
    - BASE (**B**asic **A**vailability, **S**oft state, **E**ventual consistency)

Ref: Consistency Models of NoSQL Databases - https://www.mdpi.com/1999-5903/11/2/43

ACID (1)

- ACID is a database design principle related to transaction management
  - ✓ Atomicity
  - ✓ Consistency
  - ✓ Isolation
  - ✓ Durability
- ACID is the traditional approach to database transaction management as it is leveraged by relational database management systems

ACID (2)

- Atomicity
  - ✓Ensures that all operations will always succeed or fail completely
  - ✓No partial transactions

- Consistency
  - ✓Ensures that only **data that conforms to the constraints of database schema** can be written to the database
  - ✓Database that is in consistent state will remain in consistent stage following a successful transaction.

- Isolation
  - ✓Ensures that results of transaction are not available to other operations until it is complete.
  - ✓Critical for concurrency control.

- Durability
  - ✓Ensures that results of transaction are permanent
  - ✓Once transaction is committed , it can not be rolled back.

CAP – In Distributed Systems

- CAP stands for Consistency (C) , Availability (A) and Partition Tolerance (P)
- Triple constraint related to the distributed database systems

- Consistency
  - ✓ A read of a data item from any node indicates that the same data is present across multiple nodes
- Availability
  - ✓ A read/write request will always be acknowledged in form of success or failure in reasonable time
- Partition tolerance
  - ✓ System can continue to function when communication outages split the cluster into multiple silos and can still service read/write requests

split brain problem

Consistency levels in distributed systems

- What is consistency in distributed systems
  - Each replica Node has same view of data at a given time
  - Each read request gets most recent values of write
  - Refers to the rules related to making a concurrent, distributed system appear like a single centralized system

Types of consistency

1. Eventual consistency
2. Causal consistency
3. sequential consistency
4. Strict Consistency / Linearizability

Refer: http://dbmsmusings.blogspot.com/2019/07/overview-of-consistency-levels-in.html

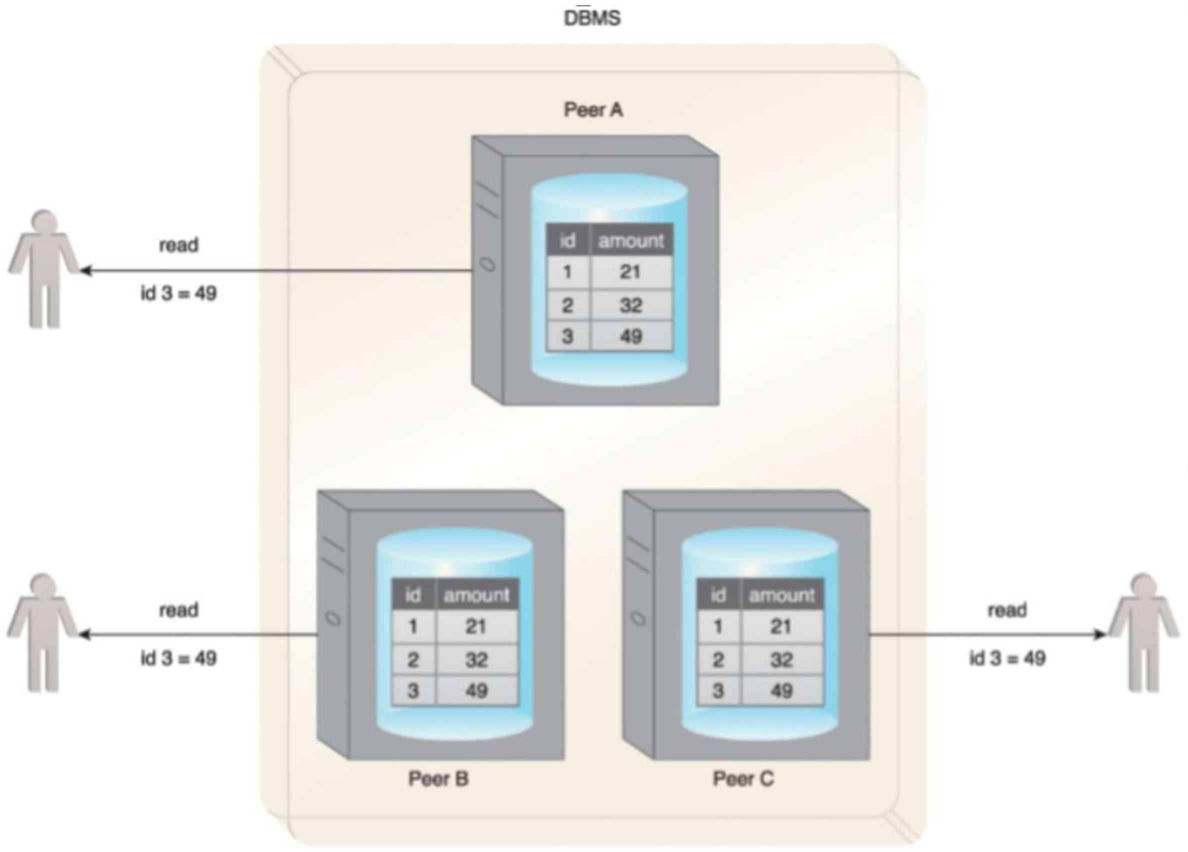
Levels of Consistency (1)

- **Eventual consistency**
  - Weakest form of consistency
  - All replicas will eventually return the same value for Read requests - If there are no writes for "some time" then all nodes will eventually agree on a latest value of the data item
  - Ensures High Availability
  - Eg:- DNS and Cassandra uses Eventual consistency
- **Causal consistency**
  - Weak consistency, but stronger than eventual consistency
  - Preserves the order of causality-related (dependent) operations
  - Does not ensure ordering of operations that are non-causal
  - Popular and useful model where only causally connected writes and reads need to be ordered. So if a write of a data item (Y) happened after a read of same or another data item (X) in a node then all nodes must observe write X before write to Y.
  - Eg: Comments and their replies on social media
- **Sequential**
  - Stronger than causal consistency
  - All writes across nodes for all data items are globally ordered. All nodes must see the same order. But does not need real-time ordering.
  - Preserves the ordering specified by each client's program
  - Does not ensure writes are visible instantaneously or in the same order as they occurred
  - Eg: FB posts

Levels of Consistency (2)

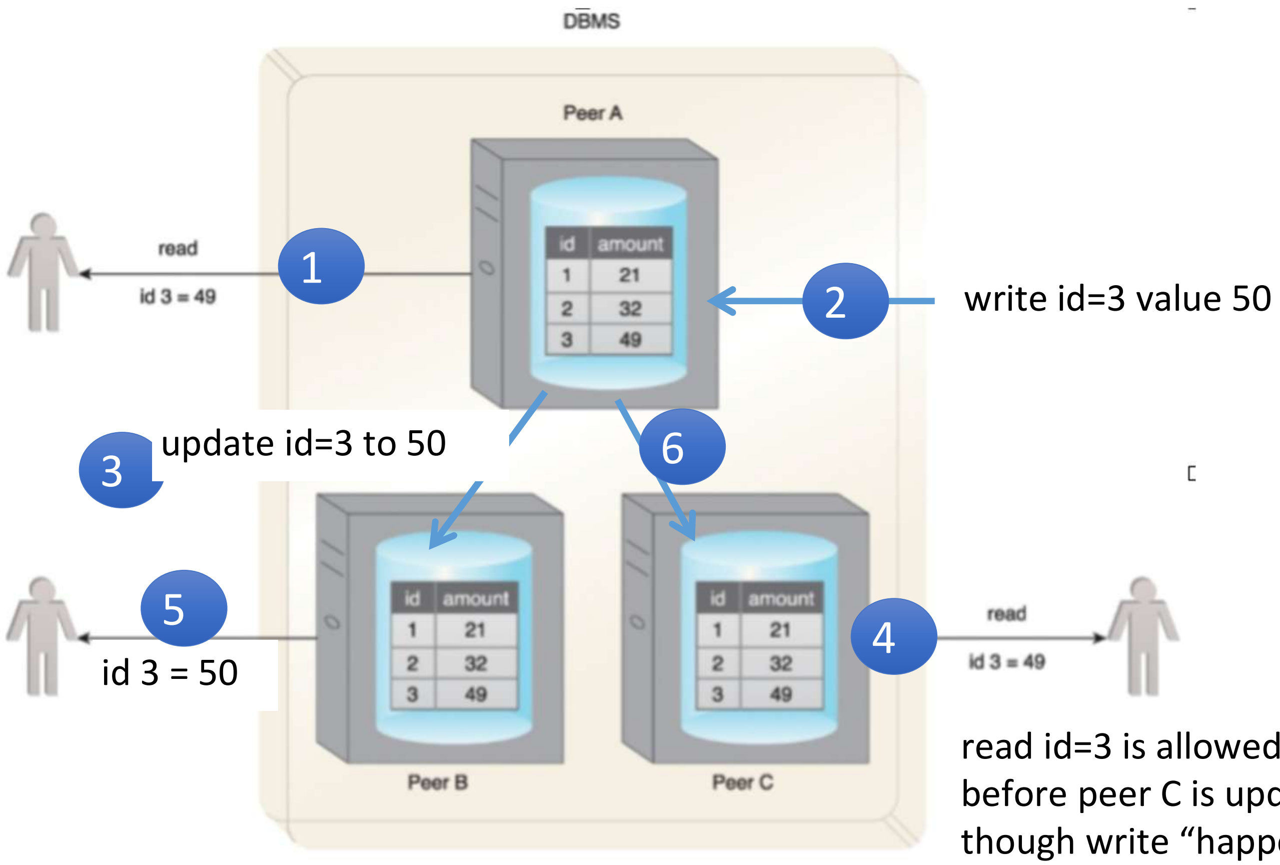
- **Strict consistency**
  - Strongest consistency model
  - Read request from any replicas get the latest write value
  - Requires real-time line ordering of all writes - assumes actual write time can be known. So “reads” read the latest data in real time across processors.
  - Eg: Updates on LDAP
- **Linearizability**
  - Linearizability is very similar to strict consistency
  - Linearizability model acknowledges that there is a period of time that occurs between when an operation is submitted to the system, and when the system responds with an acknowledgement that it was completed
  - Acknowledges that write requests take time to write all copies - so does not impose ordering within overlapping time periods of read / write
  - A linearizability guarantee does not place any ordering constraints on operations that occur with overlapping start and end times.

C: Consistency



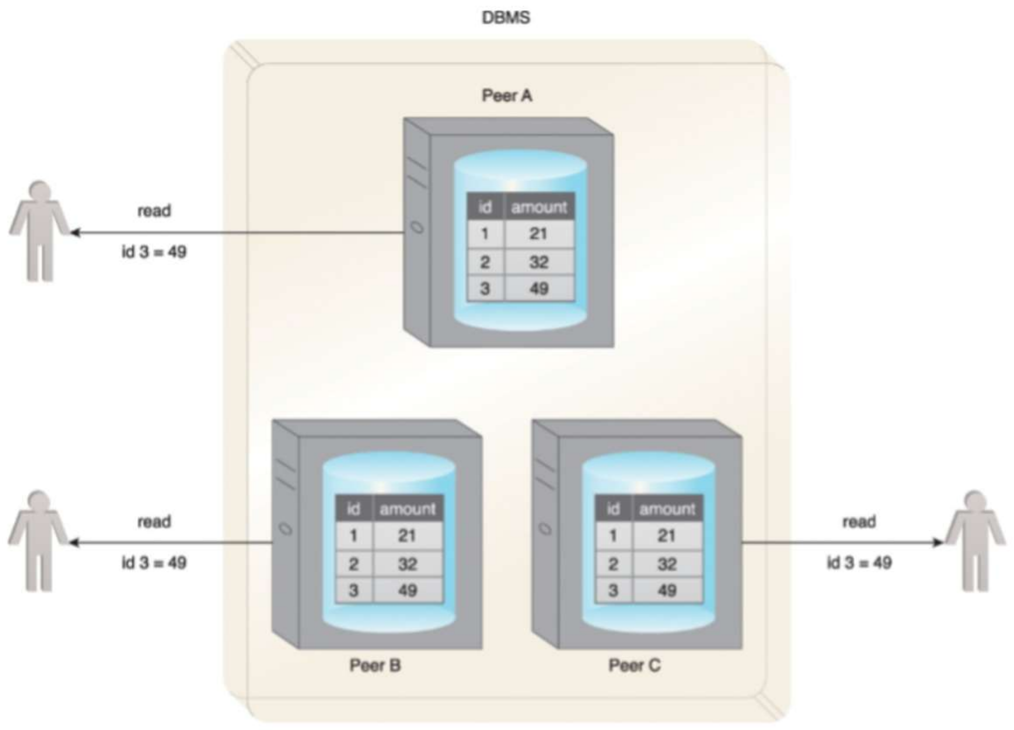
All users get the same value for the amount column even though different replicas are serving the record

When can it be in-consistent



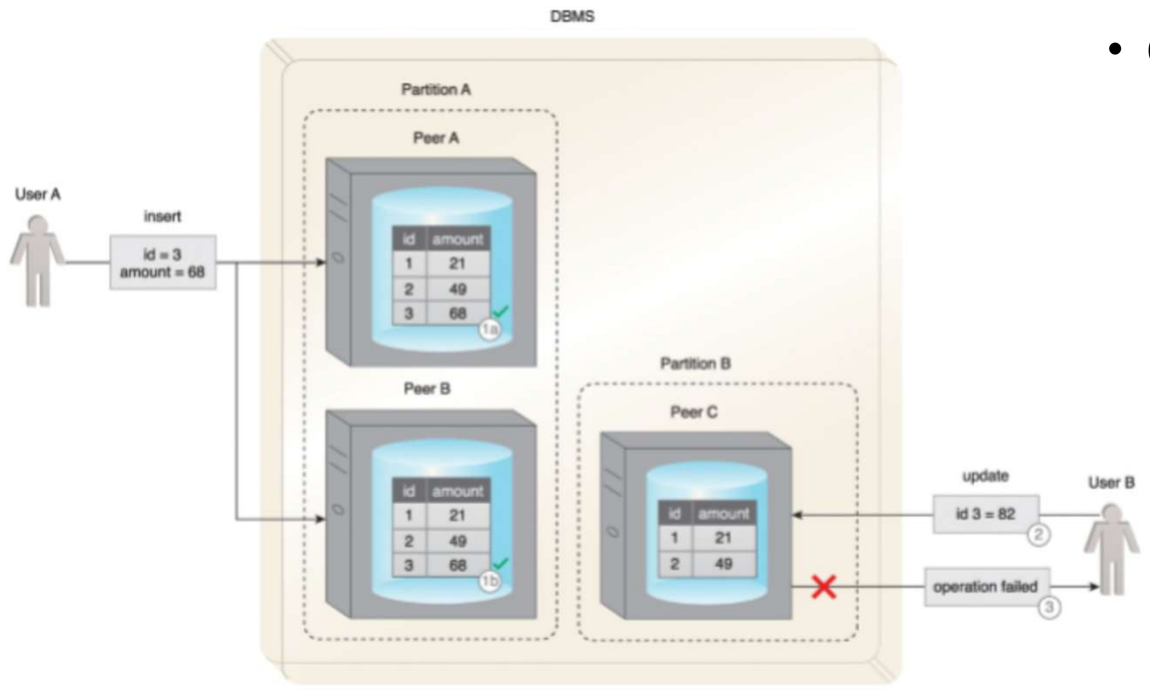
read id=3 is allowed
before peer C is updated even
though write “happens before”
read

A: Availability



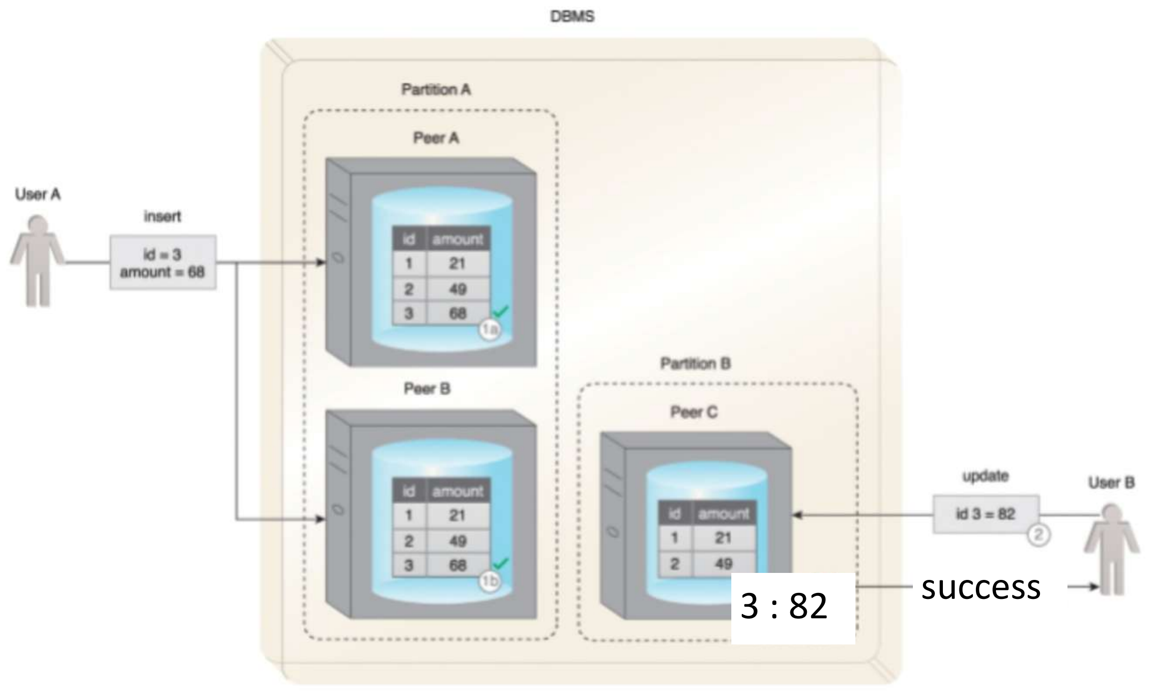
- A request from a user to a peer always responds with a success or failure within a reasonable time.
- Say Peer C is disconnected from the network, then it has 2 options
    - Available: Respond with a failure when any request is made, or
    - Unavailable: Wait for the problem to be fixed before responding, which could be unreasonably long and user request may time out

P: Partition tolerance (1)



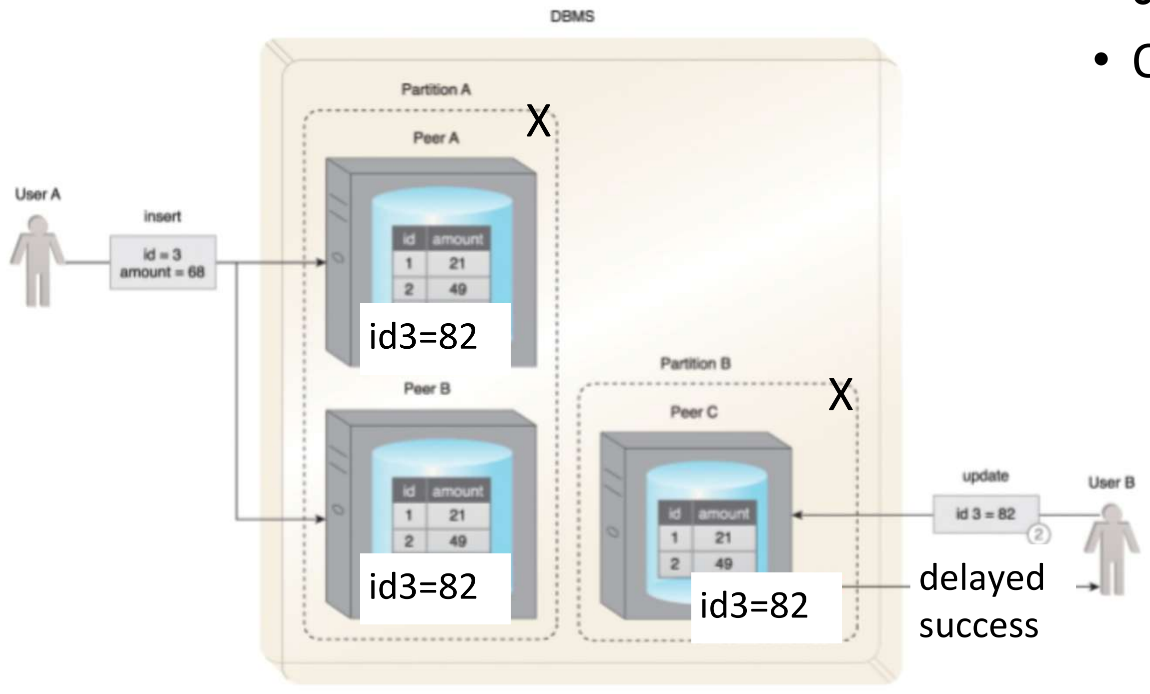
- A network partition happens and user wants to update peer C
- Option 1:
  - Any access by user on peer C leads to failure message response
  - System is available because it comes back with a response
  - There is consistency because user's update is not processed
  - But there is no partition tolerance
  - <u>C and A no P</u>

P: Partition tolerance (2)



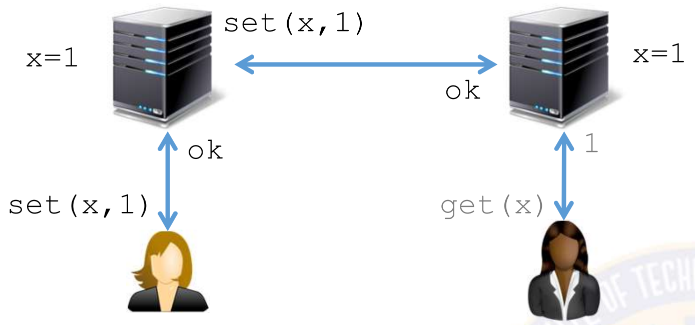
- A network partition happens and user wants to update peer C
- Option 2:
  - Peer C records the update by user with success
  - System is still available and now partition tolerant.
  - But system becomes inconsistent
  - P and A but no C

P: Partition tolerance (3)



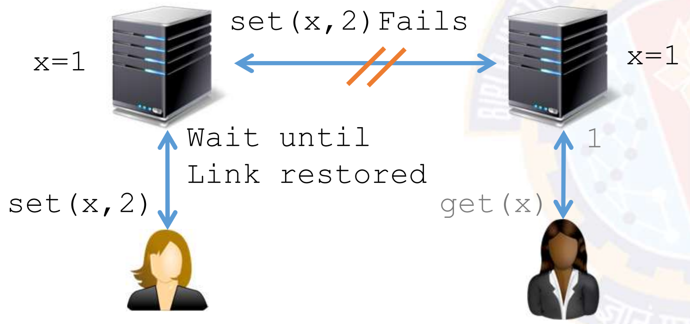
- A network partition happens and user wants to update peer C
- Option 3:
  - User is made to wait till partition is fixed (could be quite long) and data replicated on all peers before a success message is sent
  - System appears unavailable to user though it is partition tolerant and consistent
  - P and C but no A

Simple Illustration of CAP



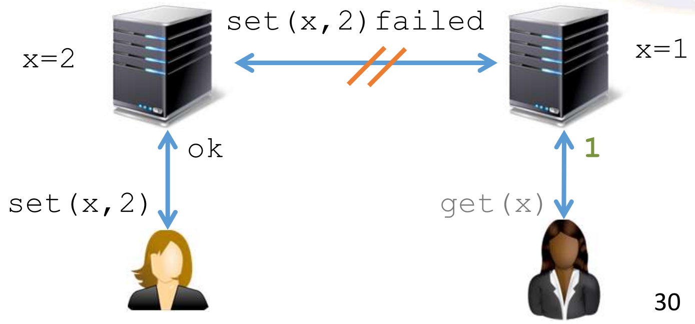
CA

Consistency, **A**vailability



CP

Consistency, **P**artition tolerance



AP

Availability, **P**artition tolerance

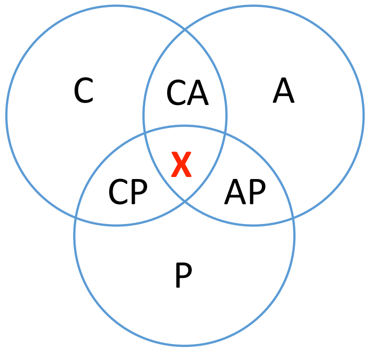
Topics for today

- Big Data Analytics lifecycle
- Consistency, Availability, Partition tolerance
- **CAP theorem**
- NoSQL Introduction
- Classification of NoSQL databases

CAP Theorem (Brewer’s Theorem)

A distributed data system, running over a cluster, can only provide ***any two*** *of the three properties C, A, P but not all.*

- So a system can be
  - CA or AP or CP but cannot support CAP
- In effect, when there is a partition, the system has to decide whether to pick consistency or availability.



CAP Theorem – Historical note

- Prof. Eric Brewer (UC Berkeley) presented it as the CAP principle in a 1999 article

– Then as an informal conjecture in his keynote at the PODC 2000 conference

- In 2002 a formal proof was given by Gilbert and Lynch, making CAP a theorem

– [Seth Gilbert, Nancy A. Lynch: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2): 51-59 (2002)]

  - https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/

– It is mainly about making the statement formal; the proof is straightforward

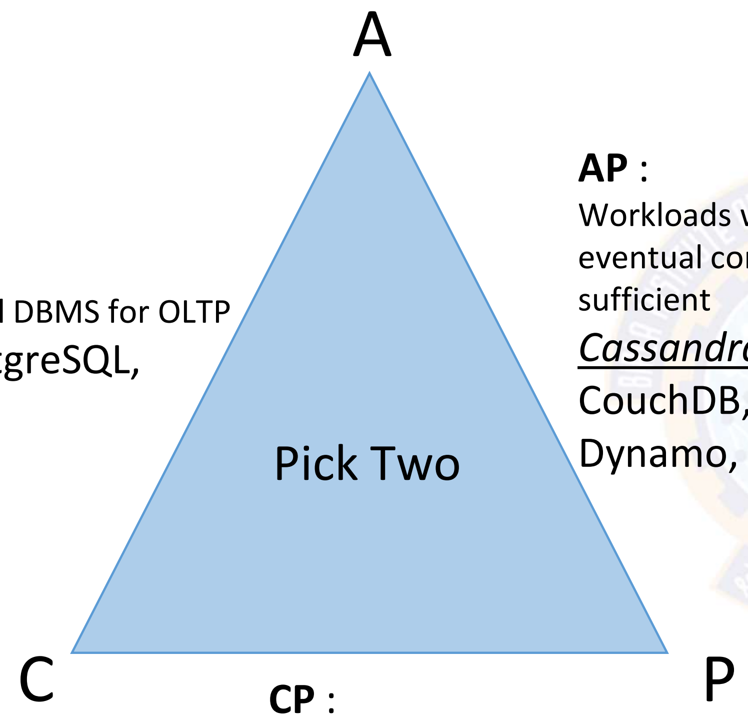
Consistency clarification

- C in ACID is different from C as in CAP Theorem
- C in ACID of RDBMS is based on OLTP systems
  - A broader concept at a data base level for a <u>transaction involving multiple data items</u>
  - Ensures that only data that conforms to the constraints of database schema are written to the database
  - Database that is in consistent state will remain in consistent stage following a successful transaction
- C as in the context of CAP Theorem for most NoSQL systems
  - A read of a data item from any node indicates that the same data is present across multiple nodes
  - Applies to <u>ordering of operations for a single data item</u> AND not a transaction involving multiple data items
  - So, it is a <u>strict subset</u> of C in ACID
- Typically support of full ACID semantics for NoSQL systems defeats the purpose as it involves having a single transaction manager that becomes a scale bottleneck for large number of partitions and replicas

Importance of CAP Theorem

- The future of databases is **distributed** (Big Data Trend, etc.)
- CAP theorem describes the **trade-offs** involved in distributed systems
- A proper understanding of CAP theorem is essential to **make decisions** about the future of distributed database **design**
- Misunderstanding can lead to **erroneous or inappropriate** design choices

Database options



CA :
Traditional DBMS for OLTP
e.g. PostgreSQL, MySQL

AP :
Workloads where eventual consistency is sufficient
Cassandra, CouchDB, Dynamo,

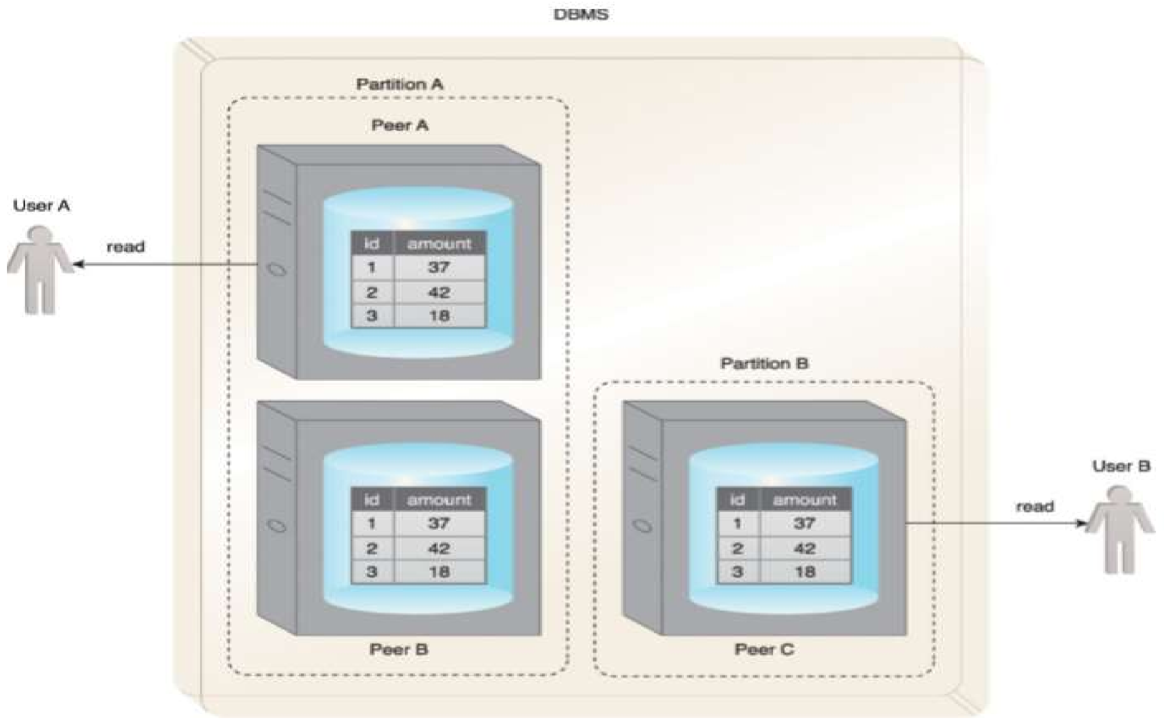
CP :
Can provide consistency with scale.
HBase, *MongoDB*, Redis

- Different design choices are made by Big Data DBs
- Faults are likely to happen in large scale systems
- Provides flexibility depending on use case to choose C, A, P behavior mainly around consistency semantics when there are faults

BASE

- BASE is a database design principle based on CAP theorem
- Leveraged by AP database systems that use distributed technology
- Stands for
  - ✓ Basically Available (BA)
  - ✓ Soft state (S)
  - ✓ Eventual consistency (E)
- It favors availability over consistency
- Soft state - Inconsistency (stale answers) allowed
- Eventual consistency - If updates stop, then after some time, consistency will be achieved
- Soft approach towards consistency allows to serve multiple clients without any latency albeit serving inconsistent results
- Not useful for transactional systems where lack of consistency is concern
- Useful for write-heavy workloads where reads need not be consistent in real-time, e.g. social media applications, monitoring data for non-real-time analysis etc.
- BASE Philosophy: best effort, optimistic, staleness and approximation allowed
- ACID to BASE Transformation - https://www.datasciencecentral.com/acid-to-base-transformation/

BASE – Basically Available



User A and User B receive data despite the database being partitioned by a network failure.

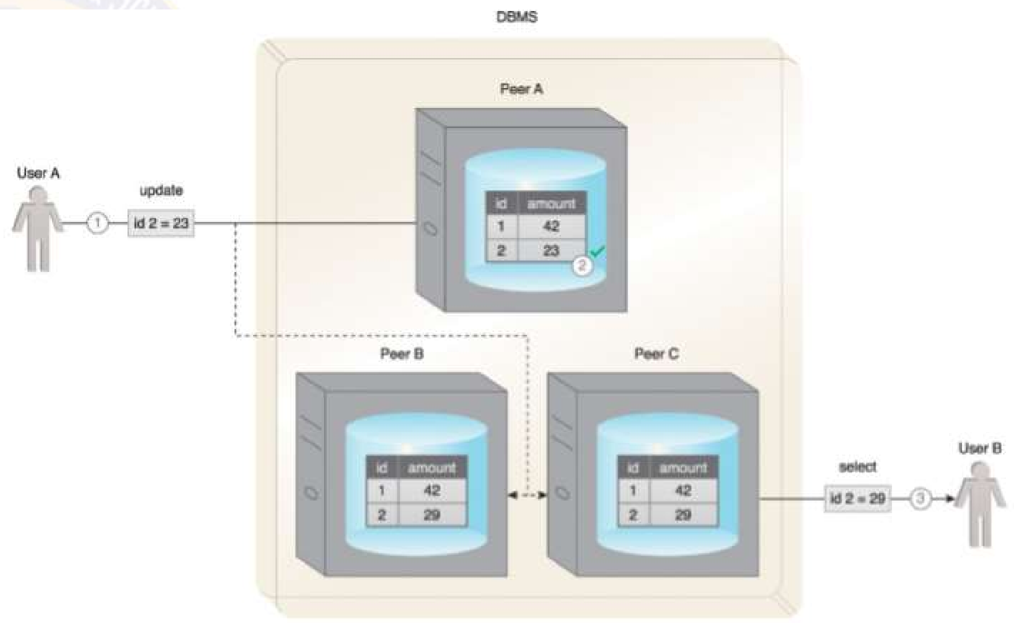
Database will always acknowledge a client’s request, either in form of requested data or a failure notification

BASE – Soft State

- Database may be in inconsistent state when data is read, thus results may change if the same data is requested again
  - ✓ Because data could be updated for consistency, even though no user has written to the database between two reads

- User A updated record on node A
- Before the other nodes are updated,

  User B requests same record from C
- Database is now in a soft state and
- Stale data is returned to User B



An example of the soft state property of BASE is shown here.

BASE – Eventual Consistency

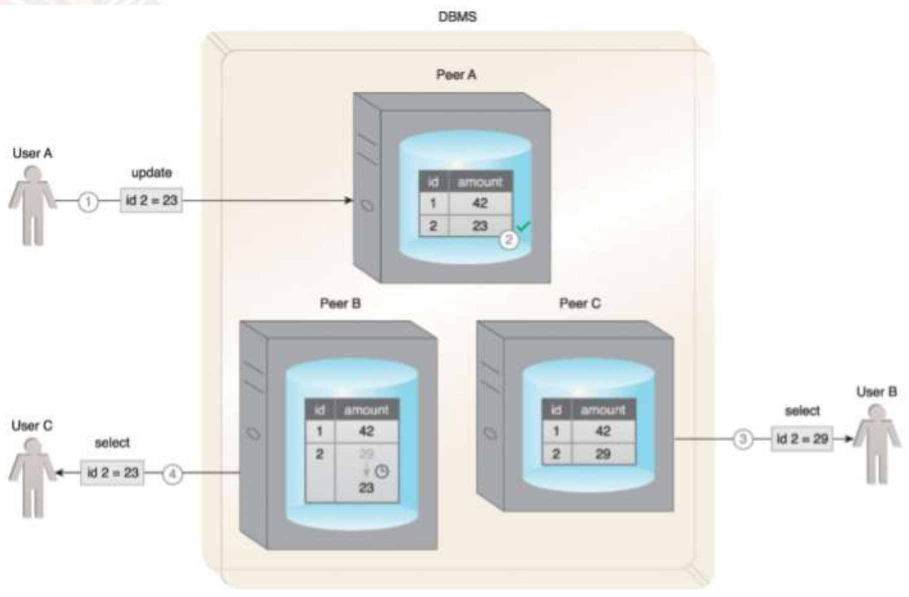
- State in which reads by different clients, immediately following a write to database, may not return consistent results
- Database only attains consistency once the changes have been propagated to all nodes
- <u>While database is in the process of attaining the state of eventual consistency, it will be in a soft state</u>

✓ User A updates a record

✓ Record only gets updated at node A, but before other peers can be updated, User B requests data

✓ Database is now in soft state, stale data is returned to User B from peer C

✓ Consistency is eventually attained, User C gets correct value

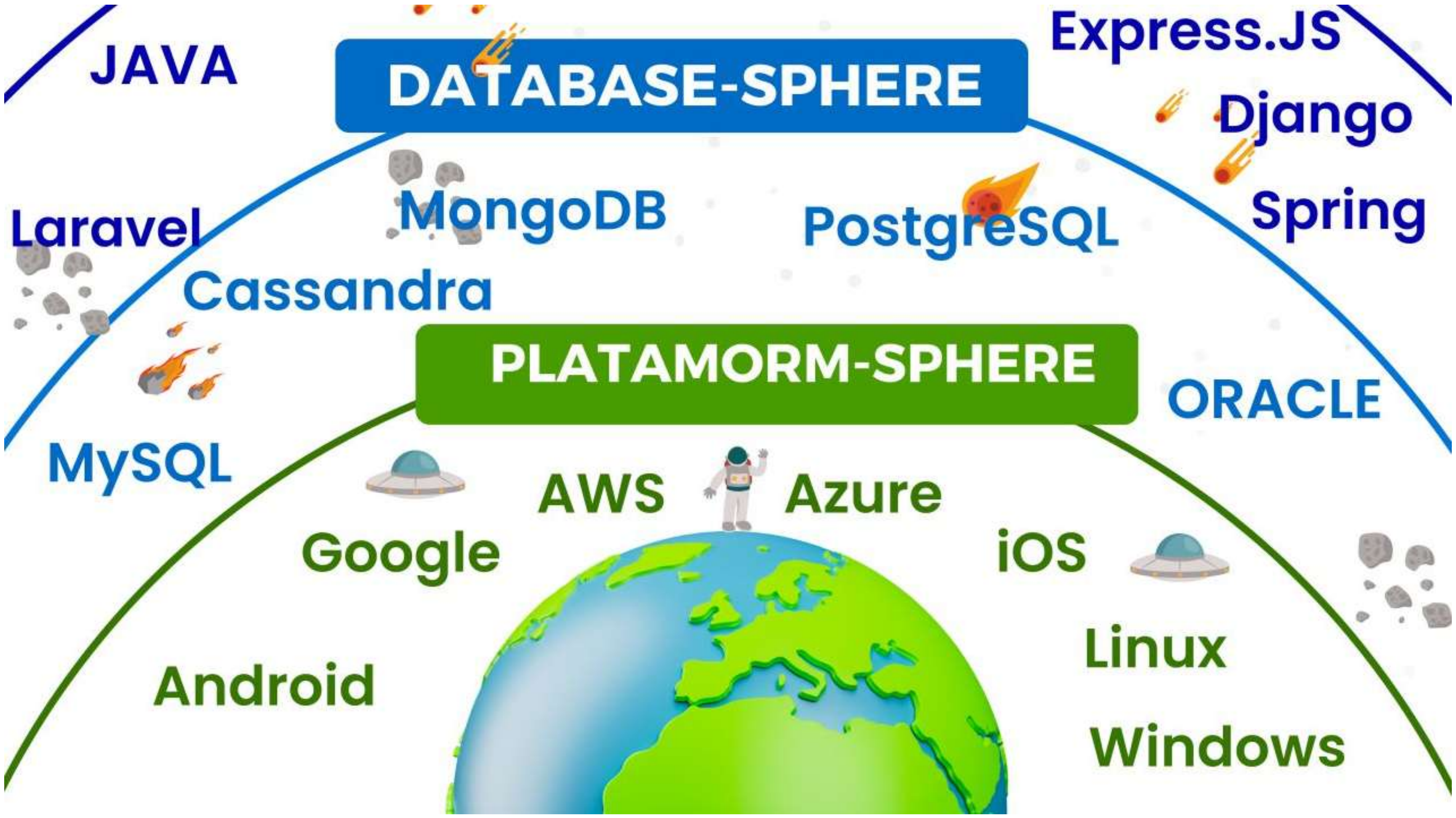


An example of the eventual consistency property of BASE.

Topics for today

- Big Data Analytics lifecycle
- Consistency, Availability, Partition tolerance
- CAP theorem
- **NoSQL Introduction**
- Classification of NoSQL databases

Database Sphere



What is NoSQL Database ?

- NoSQL databases, also known as "Not Only SQL" databases, are a type of database that do not use traditional SQL (Structured Query Language) for storing and manipulating data
- They are designed to handle large amounts of unstructured, semi-structured, or polymorphic data and are often used for big data, real-time data processing, and cloud-based applications
- NoSQL databases use a distributed architecture, allowing them to scale horizontally across multiple servers or nodes, making them ideal for handling high levels of concurrency and data volume

https://nosql-database.org/

What is NoSQL ?

- Coined by Carlo Strozzi in 1998
  - ✓Lightweight, open source database without standard SQL interface
- Reintroduced by Johan Oskarsson in 2009
  - ✓Non-relational databases
- Characteristics
  - ✓Not Only SQL
  - ✓Non-relational
  - ✓Schema-less
  - ✓Loosen consistency to address scalability and availability requirements in large scale applications
  - ✓Open source movement born out of web-scale applications
  - ✓Distributed for scale
  - ✓Cluster Friendly

Data model

- Supports rich variety of data : structured, semi-structured and unstructured
- No fixed schema, i.e. each record could have different attributes
- Non-relational - no join operations are typically supported
- Transaction semantics for multiple data items are typically not supported
- Relaxed consistency semantics - no support for ACID as in RDBMS (?)
- In some cases, can model data as graphs and queries as graph traversals

How to choose the right NoSQL database?

5 questions to ask before choosing a NoSQL database

- Is NoSQL the right choice?
- Which NoSQL data model do we need?
- What is the latency requirement?
- How important are scalability and data consistency?
- How do we want to deploy it?

Link to infoworld article

Why NoSQL (1)

- RDBMS meant for OLTP systems / Systems of Record
  - Strict consistency and durability guarantees (ACID) over multiple data items involved in a transaction
  - But they have scale and cost issues with large volumes of data, distributed geo-scale applications, very high transaction volumes
- Typical web scale systems do not need strict consistency and durability for every use case
  - Social networking
  - Real-time applications
  - Log analysis
  - Browsing retail catalogs
  - Reviews and blogs
  - …

Why NoSQL (2)

- RDBMS ensure uniform structure and modelling of relationships between entities
- A class of emerging applications need granular and extreme connectivity information modelled between individual semi-structured data items. This information needs to be also queried at scale without large expensive joins.
  - Connectivity between users in a social media application: How many friends do you have between 2 hops ?
  - Connectivity between companies in terms of domain, technology, people skills, hiring : Useful for skills acquisition, M&A etc.
  - Connectivity between IT network devices: Useful for troubleshooting incidents

Choice between Consistency and Availability

- In a distributed database
  - Scalability and fault tolerance can be improved through additional nodes, although this puts challenges on maintaining consistency (C).
  - The addition of nodes can also cause availability (A) to suffer due to the latency caused by increased communication between nodes.
    - May have to update all replicas before sending success to client . so takes longer time and system may not be available during this period to service reads on same data item.
- Large scale distributed systems cannot be 100% partition tolerant (P).
  - Although communication outages are rare and temporary, partition tolerance (P) must always be supported by distributed database

- In NoSQL, generally a choice between choosing either CP or AP of CAP
- RDBMS systems mainly provide CA for single data items and then on top of that provide ACID for transactions that touch multiple data items.

CAP Theorem – Implications of CA, CP, AP

- CA- Implies single site cluster in which all nodes communicate with each other.
- CP – Implies all the available data consistent or accurate, but some data may not be available
- AP – Implies all data available, but some data returned may be inconsistent

When you select a NoSQL database in the above categories, check weather it will match with the requirements of the application.

NoSQL Characteristics

- Scale out architecture instead of monolithic architecture of relational databases
  - Cluster scale - distribution across 100+ nodes across DCs
  - Performance scale - 100K+ DB reads and writes per sec
  - Data scale - 1B+ docs in DB
- House large amount of structured, semi-structured and unstructured data
- Dynamic schemas
  - ✓ allows insertion of data without pre-defined schema
- Auto sharding
  - ✓ automatically spreads data across the number of servers
  - ✓ applications need not be aware about it
  - ✓ helps in data balancing and recovery from failure
- Replication
  - ✓ Good support for replication of data which offers high availability, fault tolerance

NoSQL Use Cases

- **Big data:** NoSQL databases are perfect for handling large amounts of data since they can scale horizontally across multiple servers or nodes and handle high levels of concurrency
- **Real-time data processing:** They are often used for real-time data processing since they can handle high levels of concurrency and support low latency
- **Cloud-based applications:** NoSQL databases are perfect for cloud-based applications since they can easily scale up/down and handle large amounts of data in distributed environment
- **Content management:** NoSQL databases are often used for content management systems since they can handle large amounts of data and support flexible data models
- **Social media:** NoSQL databases are often used for social media applications since they can handle high levels of concurrency and support flexible data models
- **Internet of Things (IoT):** These databases are often used for IoT applications since they can handle large amounts of data from a large number of devices and handle high levels of concurrency
- **E-commerce:** They are often used for e-commerce applications since they can handle high levels of concurrency and support flexible data models

NoSQL - Pros and Cons

Pros

- Cost effective for large data sets
- Easy to implement
- Easy to distribute esp across DCs
- Easier to scale up/down
- Relaxes data consistency when required
- No pre-defined schema
- Easier to model semi-structured data or connectivity data
- Easy to support data replication

Cons

- Joins between data sets / tables
- Group by operations
- ACID properties for transactions
- SQL interface
- Lack of standardisation in this space
  - Makes it difficult to port from SQL and across NoSQL stores
- Less skills compared to SQL
- Lesser BI tools compared to mature SQL BI space

SQL vs NoSQL

SQL	NoSQL
Relational database	Non relational, distributed databases
Pre-defined schema	Schema less
Table based databases	Multiple options: Key-Value, Document, Column, Graph
Vertically scalable	Horizontally scalable
Supports ACID properties	Supports CAP theorem
Supports complex querying	Relatively simpler querying
Excellent support from vendors	Relies heavily on community support

Topics for today

- Big Data Analytics lifecycle
- Consistency, Availability, Partition tolerance
- CAP theorem
- NoSQL Introduction
- **Classification of NoSQL databases**

Classification of NoSQL DBs

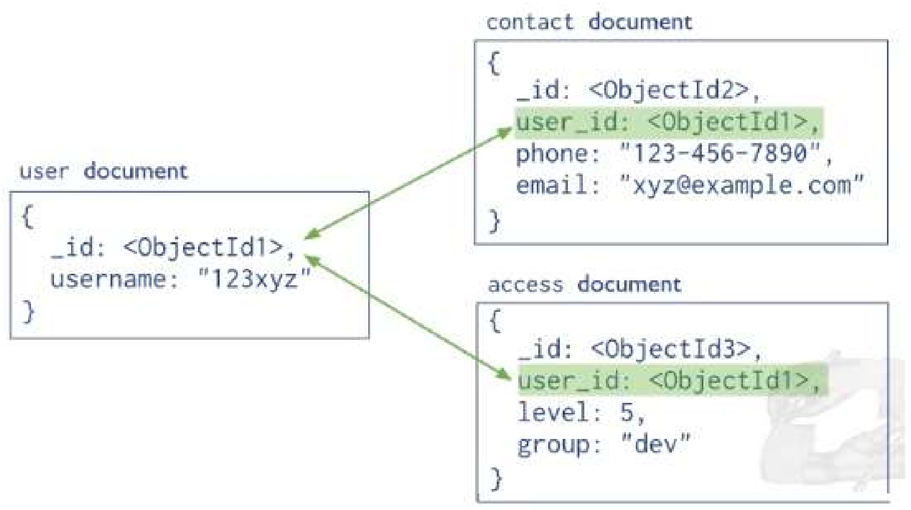
- Key – value
  - ✓ Maintains a big hash table of keys and values
  - ✓ Example : DynamoDB, Redis, Riak etc
- Document
  - ✓ Maintains data in collections of documents
  - ✓ Example : MongoDB, CouchDB etc
- Column (wide-column)
  - ✓ Each storage block has data from only one column
  - ✓ Example : Cassandra, HBase
- Graph
  - ✓ Network databases
  - ✓ Graph stores data in nodes
  - ✓ Example : Neo4j, GraphX, HyperGraphDB, Apache Tinkerpop

Classification: Document-based

- Store data in form of documents using well known formats like JSON
- Documents accessible via their id, but can be accessed through other index as well
- Maintains data in collections of documents
- Example,
    - MongoDB, CouchDB, CouchBase

- Book document :

```
{
      "Book Title" : "Fundamentals of Database Systems",
      "Publisher" : "Addison-Wesley",
      "Authors"    : "Elmasri & Navathe"
      "Year of Publication" : "2011"
}
```

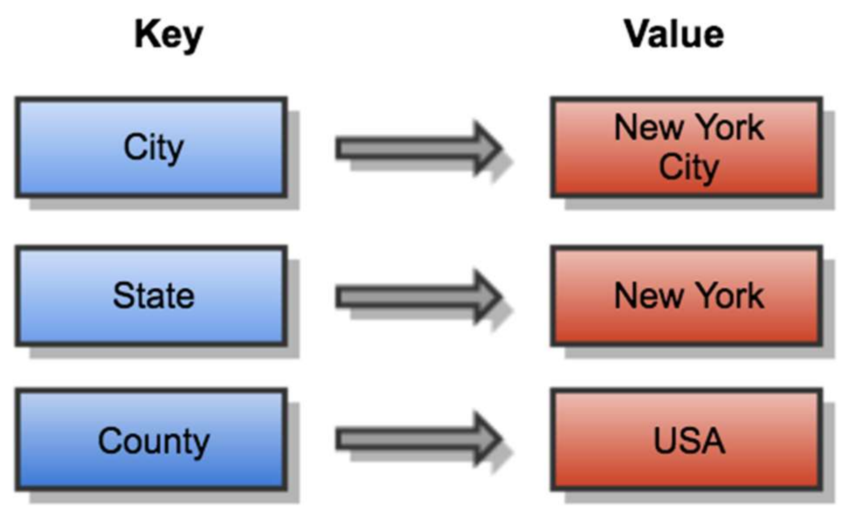


```
user document
{
  _id: <ObjectId1>,
  username: "123xyz"
}

contact document
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}

access document
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Classification: Key-Value store

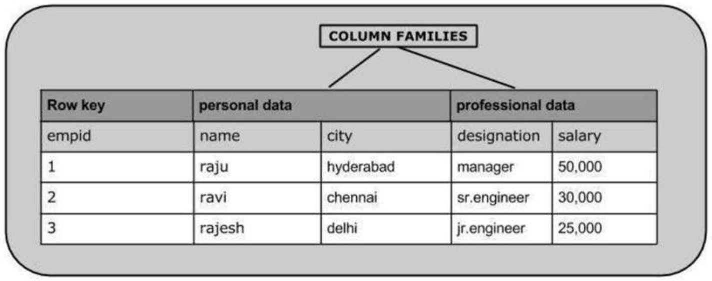
- Simple data model based on fast access by the key to the value associated with the key
- Value can be a record or object or document or even complex data structure
- Maintains a big hash table of keys and values
- For example,
  - ✓ DynamoDB, Redis, Riak

Key	Value
2014HW112220	{ Santosh,Sharma,Pilani}
2018HW123123	{Eshwar,Pillai,Hyd}

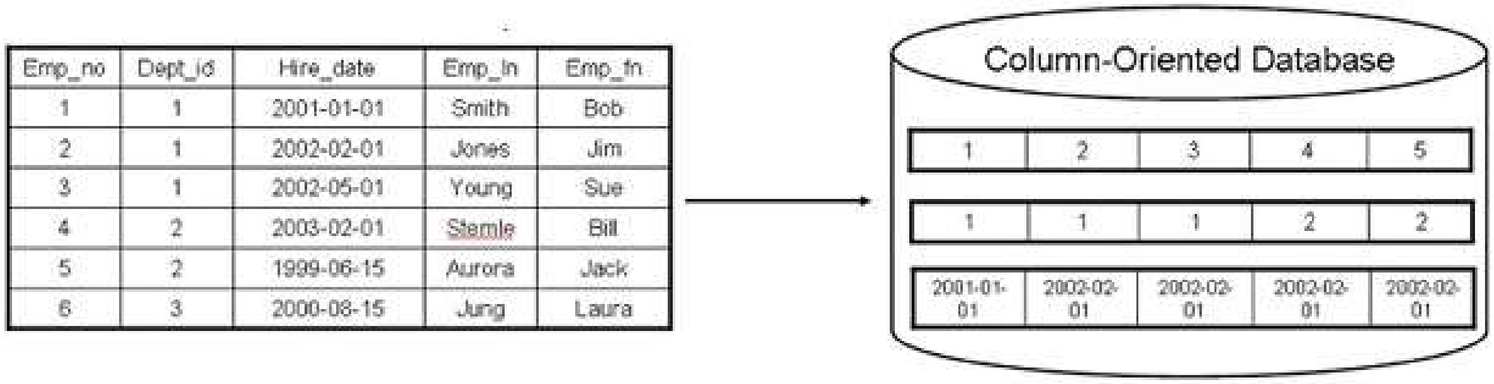


Classification: Column-based

- Partition a table by column into column families
- A part of vertical partitioning where each column family is stored in its own files
- Allows versioning of data values
- Each storage block has data from only one column
- Example,
  - ✓ Cassandra, Hbase


COLUMN FAMILIES

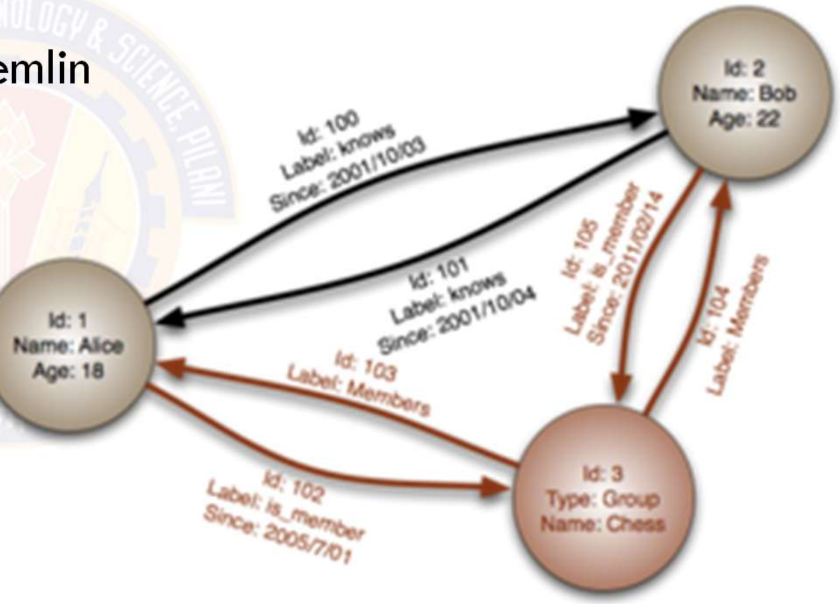
Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000



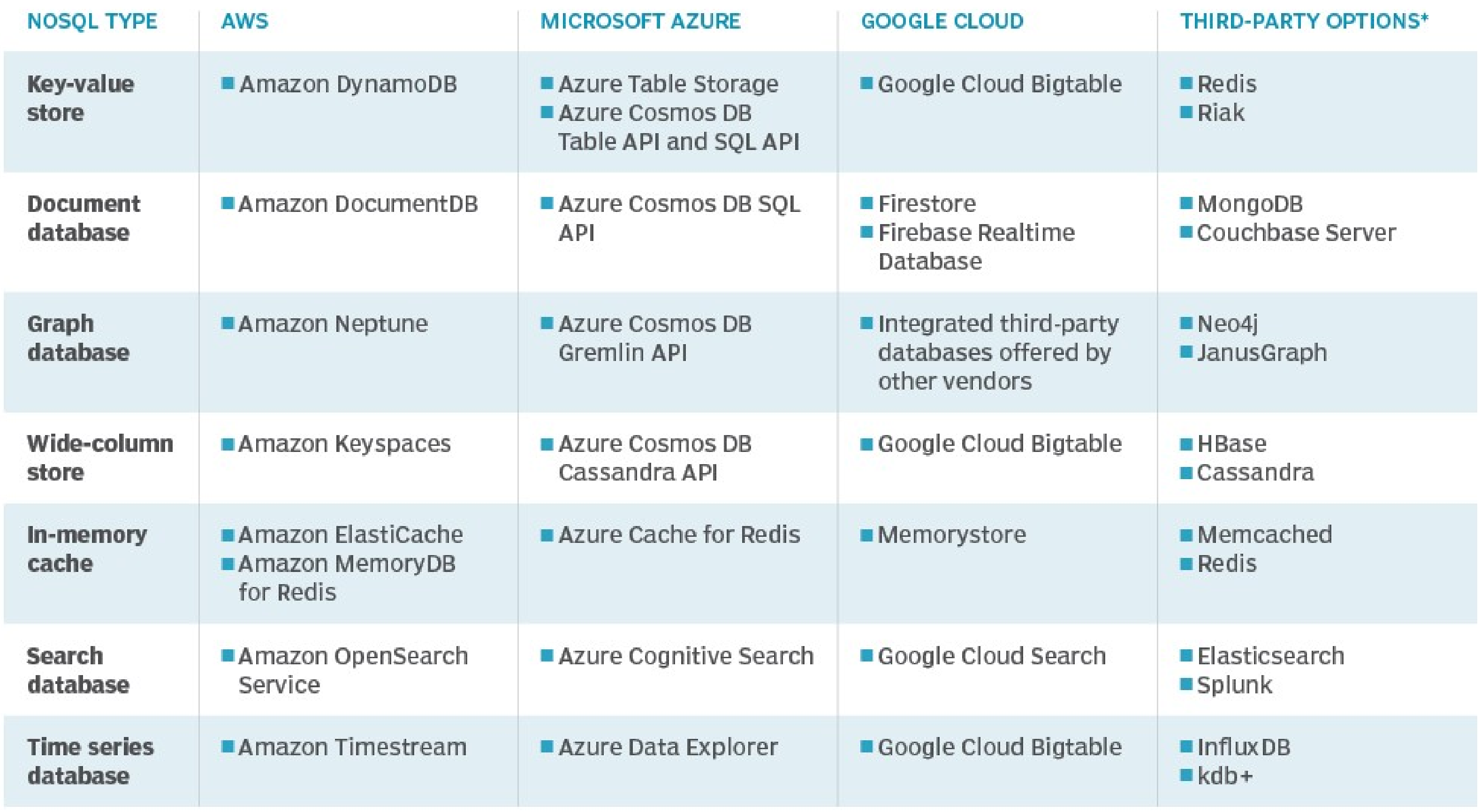
Emp_no	Dept_id	Hire_date	Emp_ln	Emp_fn
1	1	2001-01-01	Smith	Bob
2	1	2002-02-01	Jones	Jim
3	1	2002-05-01	Young	Sue
4	2	2003-02-01	Stemle	Bill
5	2	1999-06-15	Aurora	Jack
6	3	2000-08-15	Jung	Laura

Classification: Graph based

- Data is represented as graphs and related nodes can be found by traversing the edges using the path expression
- aka network database
- Graph query languages, e.g. Cypher,Gremlin
- Example
  - ✓ Neo4J
  - ✓ HyperGraphDB
  - ✓ GraphX
  - ✓ Apache TinkerPop



NoSQL Offerings in Cloud



NOSQL TYPE	AWS	MICROSOFT AZURE	GOOGLE CLOUD	THIRD-PARTY OPTIONS*
Key-value store	▪ Amazon DynamoDB	▪ Azure Table Storage ▪ Azure Cosmos DB Table API and SQL API	▪ Google Cloud Bigtable	▪ Redis ▪ Riak
Document database	▪ Amazon DocumentDB	▪ Azure Cosmos DB SQL API	▪ Firestore ▪ Firebase Realtime Database	▪ MongoDB ▪ Couchbase Server
Graph database	▪ Amazon Neptune	▪ Azure Cosmos DB Gremlin API	▪ Integrated third-party databases offered by other vendors	▪ Neo4j ▪ JanusGraph
Wide-column store	▪ Amazon Keyspaces	▪ Azure Cosmos DB Cassandra API	▪ Google Cloud Bigtable	▪ HBase ▪ Cassandra
In-memory cache	▪ Amazon ElastiCache ▪ Amazon MemoryDB for Redis	▪ Azure Cache for Redis	▪ Memorystore	▪ Memcached ▪ Redis
Search database	▪ Amazon OpenSearch Service	▪ Azure Cognitive Search	▪ Google Cloud Search	▪ Elasticsearch ▪ Splunk
Time series database	▪ Amazon Timestream	▪ Azure Data Explorer	▪ Google Cloud Bigtable	▪ InfluxDB ▪ kdb+

Vendors

- Amazon
- Facebook
- Google
- Oracle

DynamoDB

amazon web services

Amazon DocumentDB

Amazon Neptune

cassandra

APACHE HBASE

redis

ORACLE®

NOSQL DATABASE

Summary

- What process steps must one should take in a Big Data Analytics project
- What is C, A, P and various options when partitions happen
- Why is this important for Big Data environment where faults may happen in large distributed systems
- CAP Theorem
- NoSQL introduction
- Classification of NoSQL databases

Next Session:

NoSQL Database - MongoDB