DSC650: Data Technology and Future Emergence
Lecture 5 : NoSQL Database

Lecturer: Dr Khairul Anwar Sedek
Computer Science Department
Faculty of Computer and Mathematical Sciences
Universiti Teknologi MARA Perlis Branch

Where we are?

Week 8 23/5/22	5. NoSQL 5.1. Structured and Unstructured Data 5.2. Taxonomy and SQL Implementation 5.3. Basic and Related Architecture: HBase, Cassandra, MongoDB and etc.	Lab Activity 6: HBase
Semester Break / Cuti Khas Perayaan 30.05.2022 – 05.06.2022		
Week 9 6/6/22	6. Searching and Indexing Big Data 6.1. Full Text Indexing and Searching 6.2. Indexing with Lucene 6.3. Distributed Searching with Elastic Search	TEST 1 Lab Activity 7: Apache Sqoop Assignment Due
Week 10 13/6/22	7. Big Data Technologies 7.1. Introduction to Hadoop 7.2. Hadoop Ecosystem Query Language for Hadoop	Lab Activity 8: Advanced Query (Hive): Project
Week 11 20/6/22	7.3. Hadoop and Amazon Cloud 7.4. Migration to Other Big Data Platform	Project
Week 12 27/6/22	8. Trend in Data Technology 8.1. Automated Data Discovery 8.2. Deep Learning 8.3. The Next Frontier	Project
Week 13 4/7/22	Group Project - Preparation	Project Exit Survey SuFo
Week 14 11/7/22	Project Presentation & Submission	Project Due
Revision Week 17 July – 24 July 2022		
Final Assessment 25 July - 14 August 2022		

Lecture 5: **NoSQL**

Structured and Unstructured Data
Taxonomy and SQL Implementation
Basic and Related Architecture: HBase, Cassandra, MongoDB and etc

At the end of the lecture, students should be able to;

- CLO1: Demonstrate an understanding on the basic **concepts and practices** of big data technology

Source:
Jorgensen, A. et al (2014) Microsoft Big Data Solutions. John Wiley & Sons.
EMC Education Services (2015) Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. John Wiley & Sons.

NoSQL Database: Simply Explained (Video)

Terminology

Terminology Used in NoSQL and RDBMS

- **RDBMS**: Partitions, Table, Row, Column (RECALL!!!)
- **NoSQL**: Shard, Collection, Document root element (JSON/XML), Aggregated, Attribute/ field

What is NoSQL Database?

- **NoSQL** Database is a **non-relational** Data Management System, that **does not require a fixed schema**.
- It avoids joins and is easy to scale.
- The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs.
- NoSQL is used for Big data and real-time web apps.
- For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

RDBMS vs. NoSQL databases

RDBMS

- Structured data with a rigid schema.
- Extract, Transform, Load (ETL) required.
- Storage in rows and columns.
- RDBMS is based on ACID transactions. ACID stands for Atomic, Consistent, Isolated and Durable.
- RDBMS Scale up when the data load increases, i.e., expensive servers are brought to handle the additional load.
- SQL server, Oracle, and MySQL are some of the examples.
- Structured Query Language is used to query the data stored in the data warehouse.
- Matured and stable. Matured indicates that it is in existence for a number of years.

NoSQL

- Structured, Unstructured, Semi- Structured data with a flexible schema.
- ETL is not required.
- Data are stored in Key/Value pairs database, Columnar database, Document database, Graph Database.
- NoSQL is based on BASE transactions.
- BASE stands for Basically available, Soft state, Eventual consistency.
- NoSQL is highly scalable at low cost. NoSQL scales out to meet the extra load. i.e., low-cost commodity servers are distributed across the cluster. MongoDB, HBase, Cassandra are some of the examples.
- Hive Query Language (HQL) is used to query the data stored in HDFS.
- Flexible, Incubation. Incubation indicates that it is in existence from the recent past.

NoSQL Database Design

NoSQL databases exhibit the BASE properties.

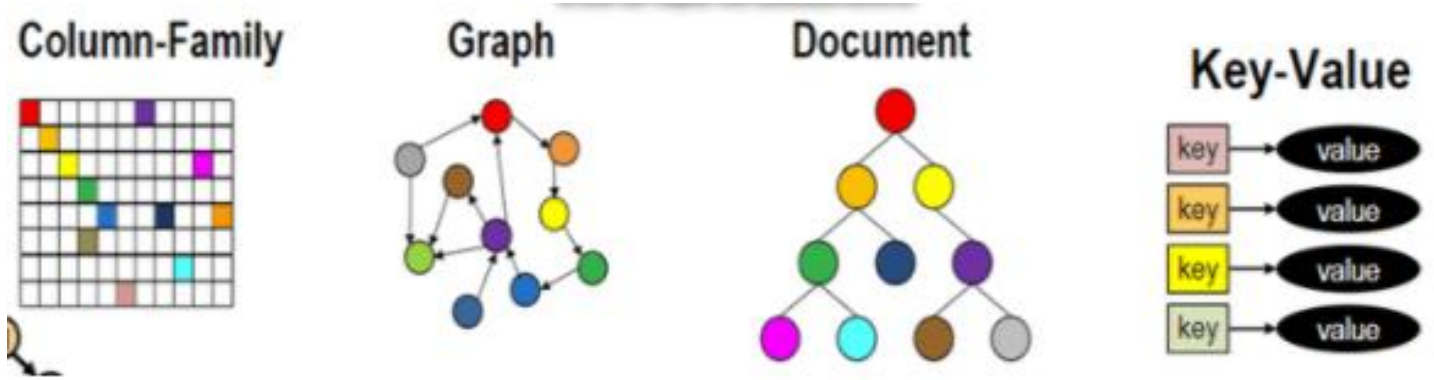
- **Basically available (availability**
  - A database is said to be basically available if the system is always available despite a network failure.
- **Soft state**
  - Soft state means database nodes may be inconsistent when a read operation is performed.
  - For example, if a user updates a record in node A before updating node B, which contains a copy of the data in node A, and if a user requests to read the data in node B, the database is now said to be in the soft state, and the user receives only stale data.
- **Eventual consistency**
  - The state that follows the soft state is eventual consistency. The database is said to have attained consistency once the changes in the data are updated on all nodes. Eventual consistency states that a read operation performed by a user immediately followed by a write operation may return inconsistent data.
  - For example, if a user updates a record in node A, and another user requests to read the same record from node B before the record gets updated, resulting data will be inconsistent; however, after consistency is eventually attained, the user gets the correct value.

NoSQL – Logical layers

- **Logical Data Model Layer**
  - with loosely typed extensible data schema (Map, Column Family, Document, Graph, etc.).
- **Data Distribution Layer**
  - ensuring horizontal scaling on multiple nodes abiding by principles of CAP theorem.
  - This comes along with necessary support for multiple data centers and dynamic provisioning (transparently adding/removing a node from a production cluster), a la Elasticity.
- **Persistence Layer** with flexibility of storing the data either in disk or memory or both; sometimes in pluggable custom stores.
- **Interface Layer** with support for various 'Non-SQL' interfaces (REST, Thrift, Language Specific APIs, etc) for data access without support for transaction.

NoSQL Database Types

- **Key/value Stores**
- **Document Stores**
- **Columnar Family Stores**
- **Graph Databases**



NoSQL Database Types: **Key/value Stores**

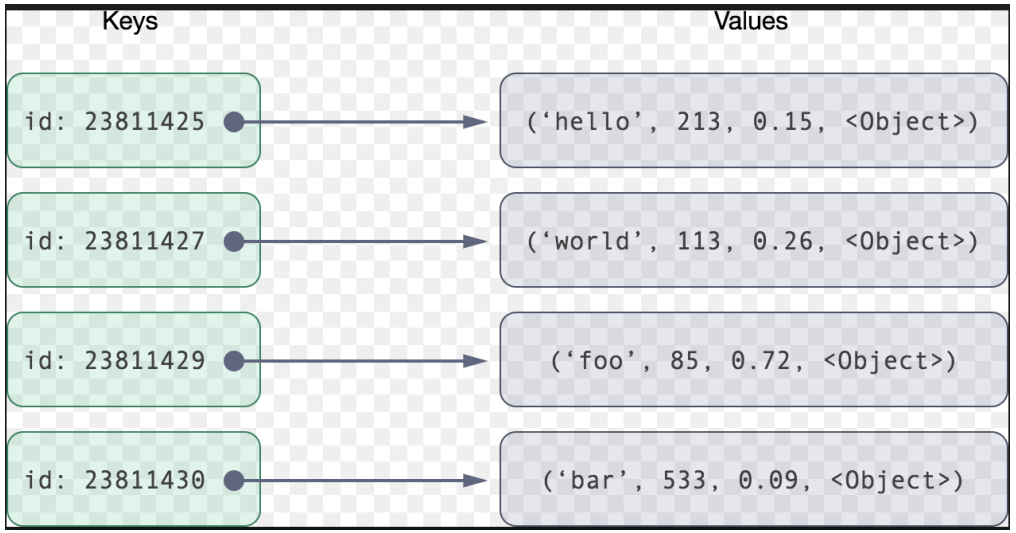
Key/value Stores:

The simplest of the NoSQL databases, key/value databases are essentially hash sets that consist of a unique key and a value that is often represented as a schema-less blob.

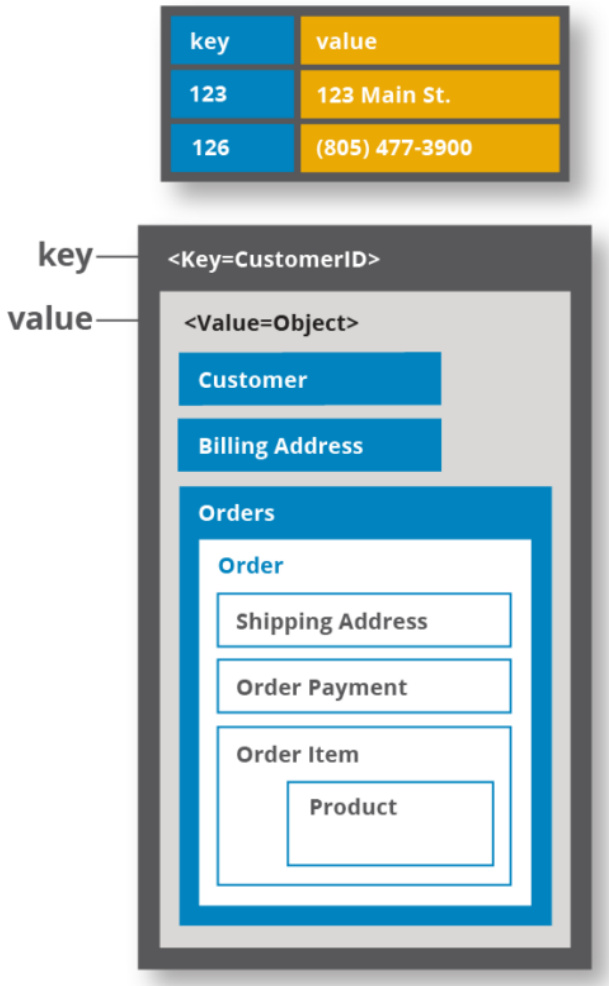
There is no stored structure of how to use the data; the client that reads and writes to a key/value store needs to maintain and utilize the logic of how to meaningfully extract the useful elements from the key and the value.

Here are some uses for key/value stores:

- Using a customer's login ID as the key, the value contains the customer's preferences.
- Using a web session ID as the key, the value contains everything that was captured during the session.

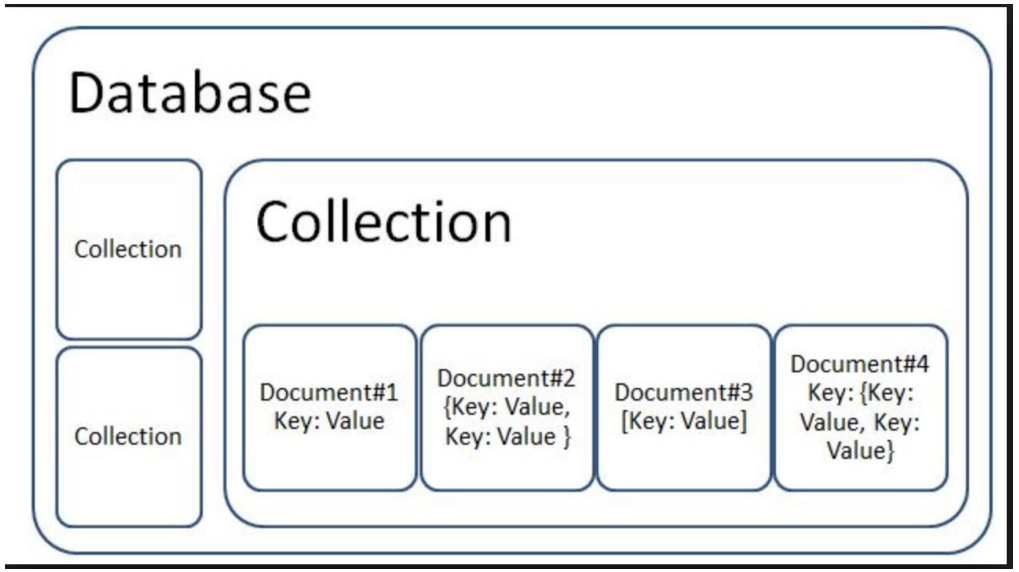


NoSQL Database Types: **Key/value Stores**:



NoSQL Database Types:
Document Stores:

- Similar to key/value databases, document databases contain structured documents (such as XML, JSON, or even HTML) in place of the schema-less blob.
- When the value of the key/value pair is a file and the file itself is self-describing (for example, JSON or XML).
- Because the document is self-describing, the document store can provide additional functionality over a key/value store.
- For example, a document store may provide the ability to create indexes to speed the searching of the documents.
  - Otherwise, every document in the data store would have to be examined.
  - These systems usually provide functionality to search within the stored documents.
- Document stores may be useful for the following:
  - Content management of web pages
  - Web analytics of stored log data




```
{
    "FirstName": "Bob",
    "Address": "5 Oak St.",
    "Hobby": "sailing"
}
```

```
<contact>
  <firstname>Bob</firstname>
  <lastname>Smith</lastname>
  <phone type="Cell">(123) 555-0178</phone>
  <phone type="Work">(890) 555-0133</phone>
  <address>
    <type>Home</type>
    <street1>123 Back St.</street1>
    <city>Boys</city>
    <state>AR</state>
    <zip>32225</zip>
    <country>US</country>
  </address>
</contact>
```

NoSQL Database Types: **Document Stores**

- **Document Stores** may contain information that would be spread across several relational tables in an RDBMS

Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

NoSQL Database Types

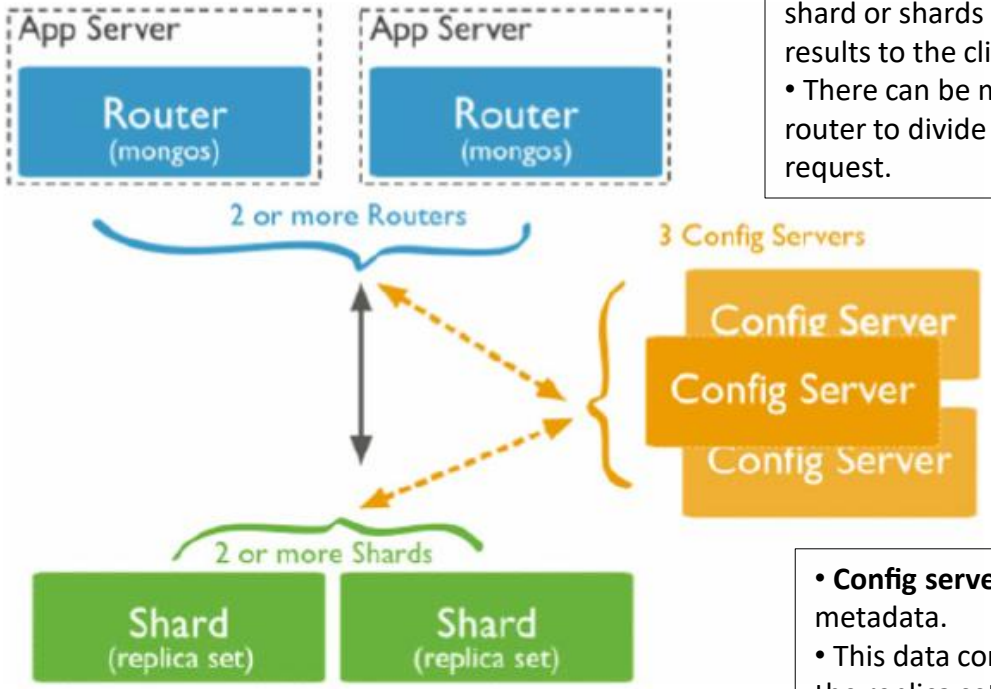
Document Stores: An example of document-oriented information could be a book database in which the key is the book title and also the value is book metadata expressed as an XML document or JSON

```
[{
    "year": 2001,
    "title": "A Programmer's Guide to ADO.NET",
    "info": {
        "author": "Mahesh Chand",
        "release_date": "2001-02-01",
        "publisher": "APress",
        "price": "44.95",
        "image_url": "ADOBook.jpg"
    }
}, {
    "year": 2003,
    "title": "GDI+ Programming",
    "info": {
        "author": "Mahesh Chand",
        "release_date": "2003-03-01",
        "publisher": "Addison Wesley",
        "price": "49.95",
        "image_url": "GDIPlusBook.jpg"
    }
}]
```

NoSQL Database Types: Document Store - MongoDB

High level processing architecture of MongoDB

• **Query Routers**, or mongos instances, interface with client applications and direct operations to the appropriate shard or shards and then returns results to the clients.
• There can be more than one query router to divide the load of the client request.



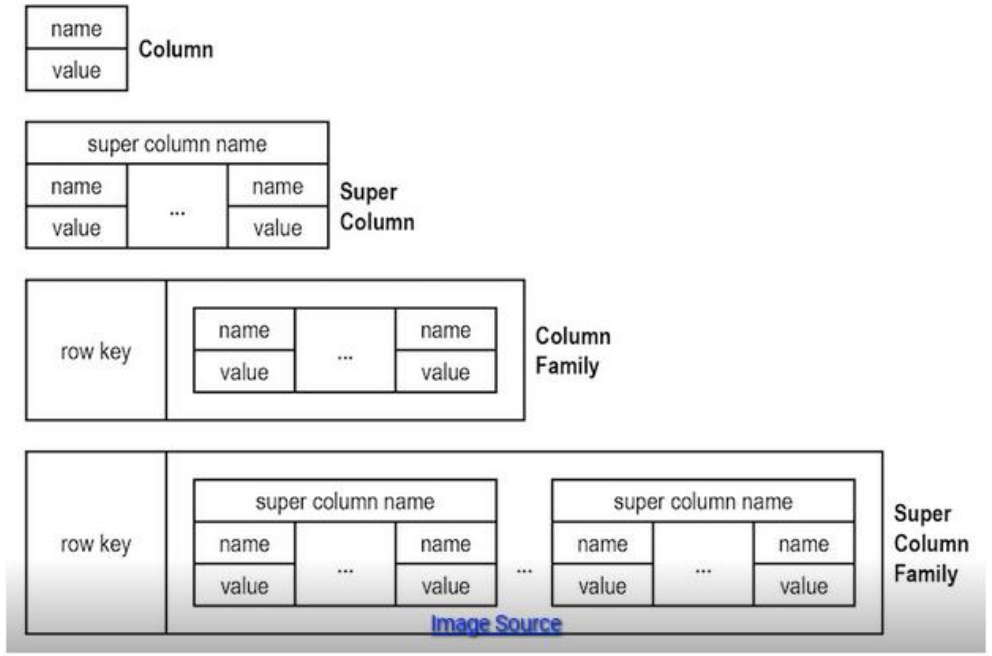
• various types of indexes for ease of data access.
• Each **shard** contains a default "_id" field serves as Primary Index.
• In addition to that secondary indices can be defined on single field, multiple fields, and arrays within the document object.

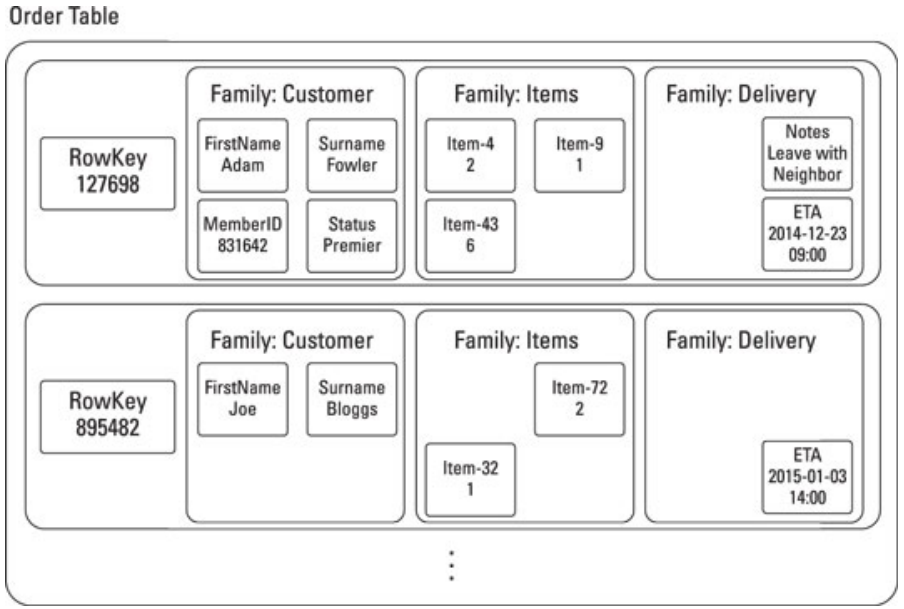
• ensures balanced sharded cluster using two background processes: the splitting and the balancing.
• Inserts and updates trigger splits.
• Balancing process also gets kicked off after addition and deletion of a Shard.

• **Config servers** store the cluster's metadata.
• This data contains a mapping of the replica sets to the shards.

NoSQL Database Types: **Columnar Family Stores**:

- Instead of storing data in a row/column approach, data in a columnar database is organized by column families which are groups of related columns.
- are useful for sparse datasets, records with thousands of columns but only a few columns have entries.
- The key/value concept still applies, but in this case a key is associated with a collection of columns.
- In this collection, related columns are grouped into column families.
  - For example, columns for age, gender, income, and education may be grouped into a demographic family.
- Column family data stores are useful in the following instances:
  - To store and render blog entries, tags, and viewers' feedback
  - To store and update various web page metrics and counters





NoSQL Database Types: **Columnar Family Stores**

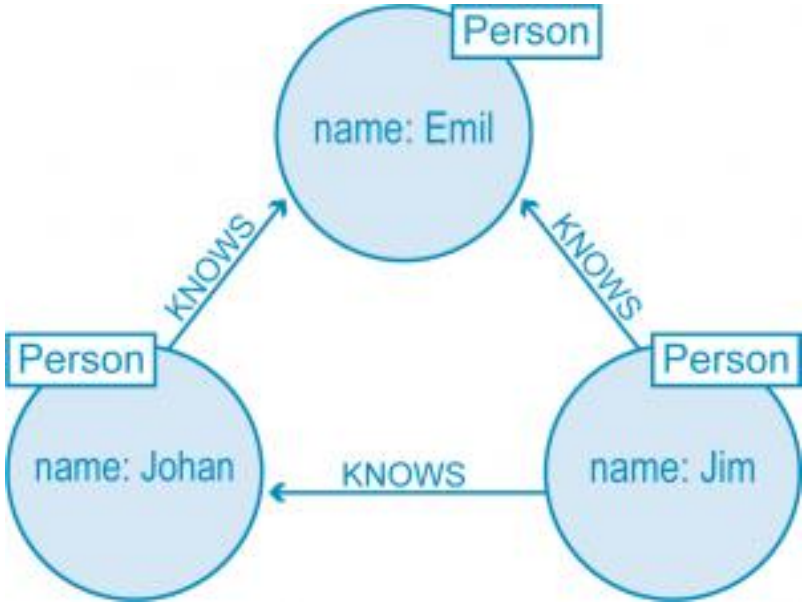
Columnar Family Stores: The following diagram shows an example with two column families, Identity and Contact Info. The data for a single entity has the same row key in each column-family.

CustomerID	Column Family: Identity
001	**First name:** Mu Bae **Last name:** Min
002	**First name:** Francisco **Last name:** Vila Nova **Suffix:** Jr.
003	**First name:** Lena **Last name:** Adamcyz **Title:** Dr.

CustomerID	Column Family: Contact Info
001	**Phone number:** 555-0100 **Email:** someone@example.com
002	**Email:** vilanova@contoso.com
003	**Phone number:** 555-0120

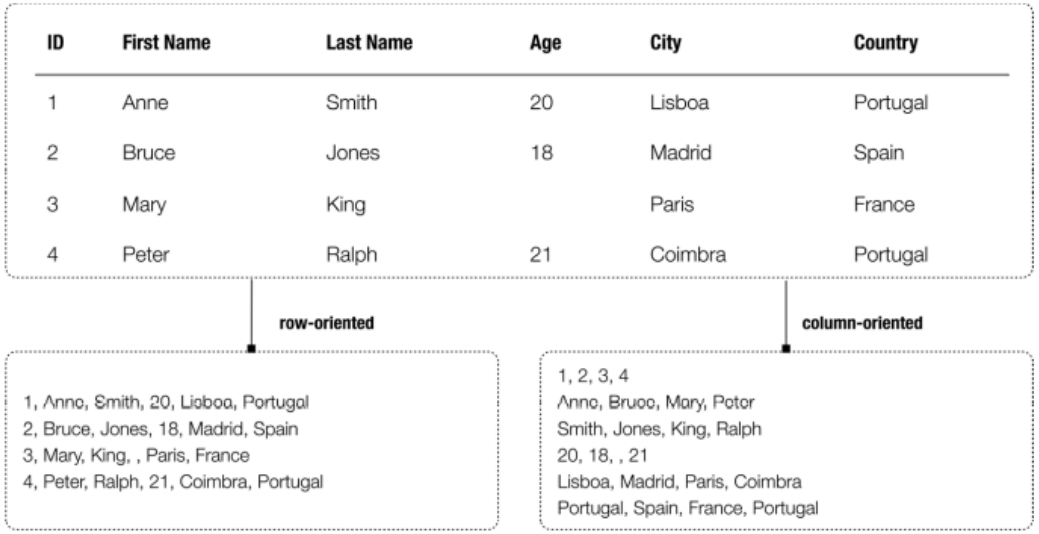
NoSQL Database Types: **Graph Databases**

- intended for use cases such as networks, where there are items (people or web page links) and relationships between these items.
- The graph database consists of entities and edges, which represent relationships between nodes.
  - The relationships between nodes can contain properties, which include items like direction of the relationship.
  - This type of NoSQL database is commonly used to traverse organization or social network data (move from one item in the network to another item in the network).
- Following are examples of graph database implementations:
  - Social networks such as Facebook and LinkedIn
  - Geospatial applications such as delivery and traffic systems to optimize the time to reach one or more destinations



Row-oriented vs Column-oriented storage

- OLTP - data retrieves less number of rows and more columns, so the row-oriented database is suitable.
- OLAP - retrieves fewer columns and more rows, so the column-oriented database is suitable.



NoSQL Data Stores Example

Category	Data Store	Website
Key/Value	Redis	redis.io
	Voldemort	www.project-voldemort.com/voldemort
Document	CouchDB	couchdb.apache.org
	MongoDB	www.mongodb.org
Column family	Cassandra	cassandra.apache.org
	HBase	hbase.apache.org/
Graph	FlockDB	github.com/twitter/flockdb
	Neo4j	www.neo4j.org

NoSQL Data Stores Example

- **Amazon's Dynamo** is an example of database that use key-value database of NoSQL.
- They use commodity hardware, standard mode of operation, loosely coupled and Service oriented architecture of hundred of services.
  - Because of using commodity hardware usage it is having scalability nature.
- Objects are stored with versioned data.
  - uses consistent hashing to dynamically partition data across the storage hosts that are present in the system at a given time.
- To maintain the consistency during updates Dynamo uses quorum-like technique (ensure that no two copies of a data item are read or written by two transactions concurrently) and a protocol for decentralized replica synchronization.

NoSQL Data Stores Example

- Project **Voldemort**
  - key-value-store which was initially developed for and still used at LinkedIn.
  - get(key), returning a value object
  - put(key, value)
  - delete(key)
- **Tokyo Cabinet** and **Tokyo Tyrant**
  - data store which is build on key-value pair of databases.
  - Tokyo cabinet is the core library of this data persistence and extracts data based on B++ tree structure or hash indexes.
  - The Tokyo suite is developed actively, well documented and give high-performance, as 1 million records can be stored in 0.7 seconds by using the hash-table engine and in 1.6 seconds by using the b-tree.

HBase

- HDFS Column-Oriented Database
- is a NoSQL database built on top of Hadoop and HDFS that provides real-time, random read/write access
- Relational designs and databases do not easily scale and cannot typically handle the volumes, variety, and velocity associated with big data environments.
- HBase is a **columnar database**, which means that instead of being organized by rows and columns, it is organized by column families, which are sets of related columns.

Hbase Database Structure

Customer

CustomerKey	FirstName	LastName	AddressKey
10	Chris	Price	901
20	Adam	Jorgensen	902
30	Donald	Duck	903

Address

AddressKey	Address1	City	State	Country	ZipCode
901	123 Main St	Tampa	FL	US	33555
902	123 Beach Ave	Jacksonville	FL	US	33666
903	123 Mickey Lane	Orlando	FL	US	33777

Traditional database structure

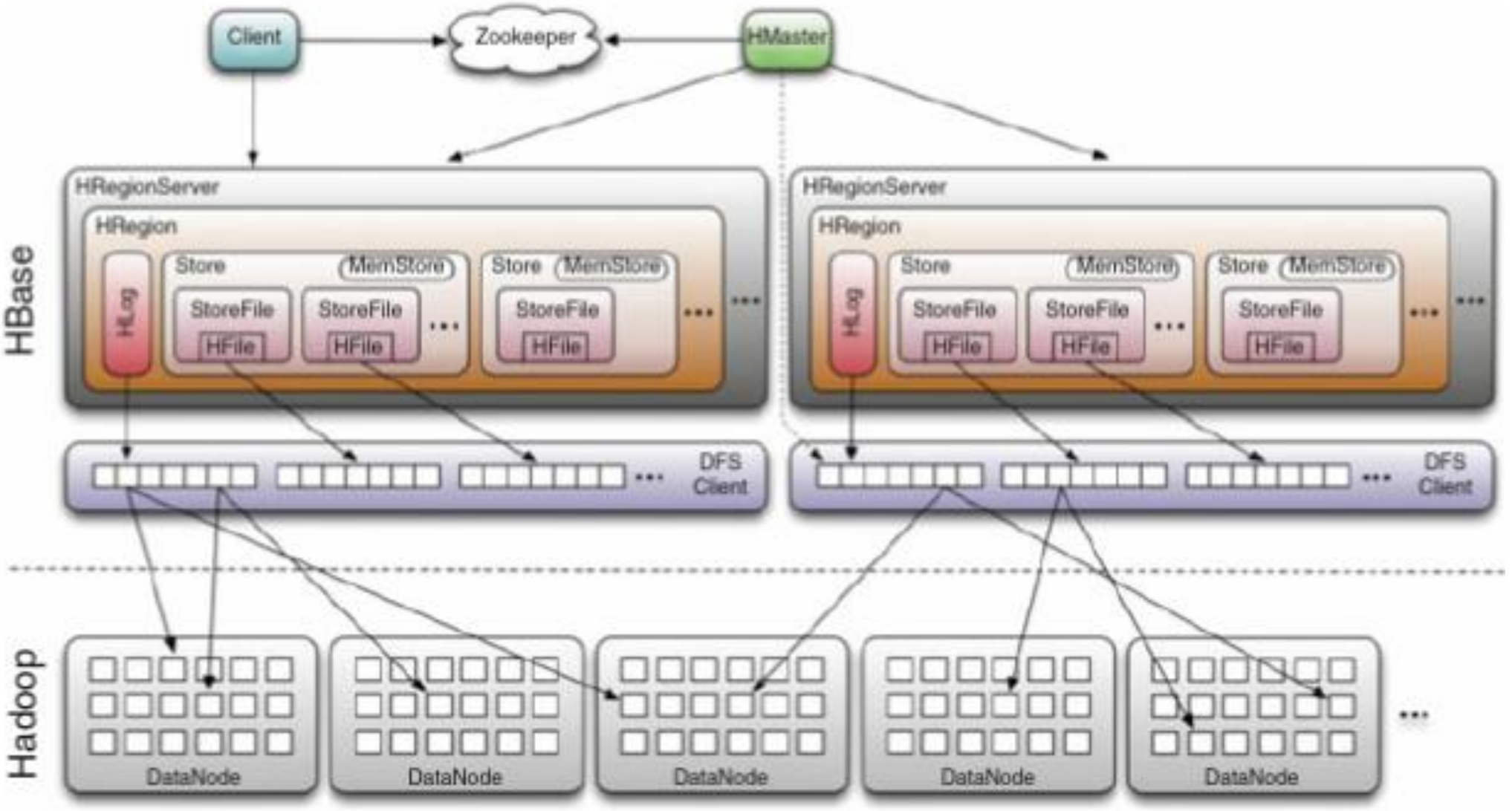
Columnar database structure

Row Key	Column Families	
Customer ID	Customer	Address
10	FirstName: Chris LastName: Price	Street1: 123 Main St. City: Tampa State: FL Country: US Zip: 33555
20	FirstName: Adam LastName: Jorgensen	Street1: 123 Beach Ave. City: Jacksonville State: FL Country: US Zip: 33666
30	FirstName: Donald LastName: Duck	Street1: 123 Mickey Circle City: Orlando State: FL Country: US Zip: 33777

Hbase - Scalability Architecture

- The MemStore holds in-memory modifications to the data.
- Each Region Server also contains a Write-Ahead Log (called HLog or WAL) for the purpose of **data durability**.
- When a Table becomes too big, it is partitioned into multiple Regions automatically by the HRegionServer.
- The mapping of Regions to Region Server is kept in a system table called .META.
- HBase cluster has one Master node, namely HMaster and multiple Region Servers namely HRegionServer.
- HMaster in the HBase is responsible for monitoring the Cluster, assigning Regions to the Region Servers, Controlling the Load Balancing, and Failover of Region Servers.
- Each Region Server contains multiple Regions, namely HRegions.
- The Data in HBase Tables are stored in these Regions.
- Each Region is identified by the start key (inclusive) and the end key (exclusive) and is made up of a MemStore and multiple StoreFiles (HFile).
- The data lives in these StoreFiles in the form of Column Families and eventually is stored in HDFS.

Hbase - Scalability Architecture



Hbase Advantages

- The columnar layout has many advantages over a relational model in the context of handling big data, including the following:
  - Can handle very large (even massive) quantities of data through a process known as sharding
  - Allows flexible data formats that can vary from row to row
  - Typically scales linearly

HBase Commands

Defining and Populating an HBase Table

- To define a table, you specify a table name and the column family or families.
- In the following example, a basic customer table with a single column family for addresses is created:

```
create 'customer', 'address‘
```

- Now, let's take a quick look at how we put data into our customer table using the put statement:

```
put 'customer', 'row01', 'address:street', '123 Main St.'
put 'customer', 'row01', 'address:city', 'Tampa'
put 'customer', 'row01', 'address:state', 'Florida'
put 'customer', 'row01', 'address:country', 'United States of America'
put 'customer', 'row01', 'address:zip', '34637‘
put 'customer', 'row02', 'address:street', ‘100 Main St.'
put 'customer', 'row02', 'address:city', ‘Arau'
put 'customer', 'row02', 'address:state', ‘Perlis'
put 'customer', 'row02', 'address:country', ‘Malaysia'
put 'customer', 'row02', 'address:zip', ‘02600‘
```

Using Query Operations

- To retrieve the data from the HBase table you just created, there are two fundamental methods available through the HBase shell. The scan command indiscriminately reads the entire contents of your table and dumps it to the console window:

```
scan 'customer'
```

- When working with a larger table, it is preferable to use a more targeted query. The get command accomplishes this:

```
get 'customer' 'row01'
```

CouchDB

- descendant of Lotus Notes, whose main developer Damien Katz worked at IBM before he later initiated the CouchDB project on his own.
- A lot of ideas from Lotus Notes can be found in CouchDB, documents, views, distribution and replication between servers and clients.
- The approach of CouchDB is to make such document database from scratch with technologies of the web space like representational State Transfer, JavaScript Object Notation (JSON) as a data interchange format, and also the ability to integrate with infrastructure elements like load balancers and caching proxies etc.
- characterized as a document database that is accessible via **a restful HTTP-interface**, containing schema-free documents in a flat address area.
- CouchDB is distributed and ready to replicate between server nodes; Similarly, as clients and servers incrementally.
- Blogs, Wikis, Social networks, Facebook apps and smaller internet sites use CouchDB as their datastore.

MongoDB

- an open source project which is also called schema free database.
- It is written in C++ and owned by 10gen Inc.
- It provides solution for issues of traditional RDBMS with scalable key-value database.
- SourceForge.net, foursquare, the New York Times, the URL-shortener bit.ly and the distributed social network DIASPORA are the main users of MongoDB..

RDBMS vs MongoDB terminology

RDBMS	MOngoDB
Database	Collection
Table	Document
Column	Field

Using MongoDB

- Inserting a document into a collection(Create)
  - insertOne()
  - insertMany()

```
db.BooksDB.insertOne({
    title:}) "Dead Silence",
    author: "S.A. Barnes",
    isbn: 1250819997,
    price: 13.99,
    available: true
```

- Find a document in database
  - `Db.BooksDB.find()`
  - `db.BooksDB.find({"title":"Day Zero"}`

- Update a Document
  - `db.BooksDB.updateOne({author: "S.A. Barnes"}, {$set:{author: "Stacey Kade Barnes"}})`
- Update many documents
  - `db.BooksDB.updateMany({author:"C. Robert Cargill"}, {$set: {author: "Christopher Robert Cargill"}})`
- Delete a document
  - `db.BooksDB.deleteOne({name:"Christopher Robert Cargill"})`
- Delete many documents
  - `db.BooksDB.deleteMany({author:"Christopher Robert Cargill"})`

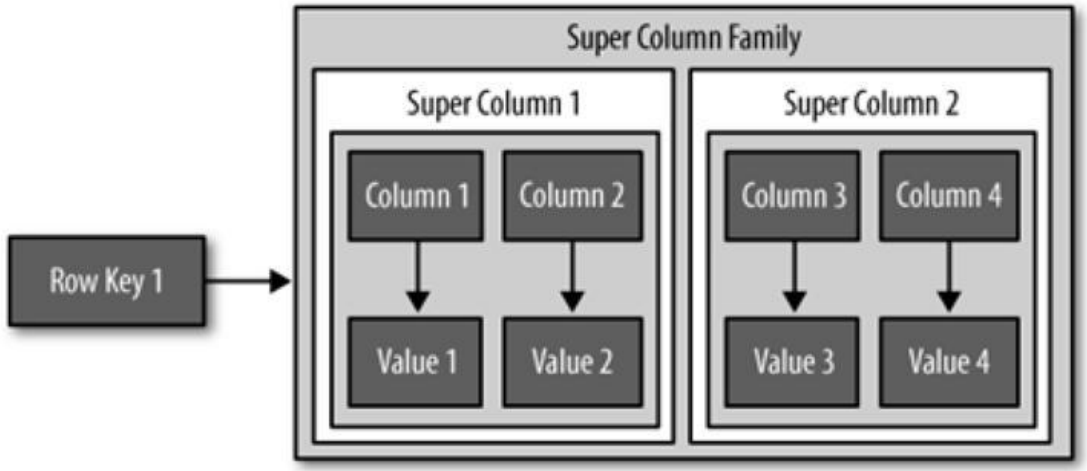
Cassandra

- Companies like Twitter, Rackspace and Digg etc initiated and are using this database.
- It also supports distributed approach for scalability issue for database.
- It also manages large amount of structure data but without supporting full relational data base.
- Facebook use this technique for its messaging system, for storage, read and forward.
- It provides facilities to store the message part and forward when it is needed, this problem is called "***Inbox search problem***".
  - With Cassandra, Facebook got scalability, availability for its messaging system.

Cassandra: Structure

- Cassandra, stored values in triplet that is (row-key, column-key, timestamp) with column-key as column-family:column.

Rows	These are identified by a string-key of arbitrary length. Atomic operations are required.
Column-family	Need to define in advance. These are not limited. Columns can be added dynamically.
Column	It is name and values that are stored in table which is identified by timestamps.
Supercolumns	Have a name and an arbitrary number of columns associated with them. Again, the number of columns per super-column may differ per row.



Cassandra Characteristics

- **Row-Oriented:** Cassandra is column-oriented NoSQL database and it represent data structure in sparse multidimensional hash table that means it makes data accessible.
- Every row is having unique key for accessing data uniquely.
  - Cassandra store data in multidimensional hash table that means there is no need to mention anything about data structure. It stores data as row oriented manner with unique key with it and perform row-oriented storage.
- **Schema-free:** Cassandra requires defining outer schemas that are key space that contain with column-families.
  - This refers as logic namespace to hold column families and its properties.
- **High Performance:** Cassandra is useful with its advantage of multiprocessor and multi core machines.
  - It can easily scale with multiple machine support with thousands of terabytes.
  - With heavy traffic of data load Cassandra perform well with availability, scalability and tolerance.
  - Adding more servers can increase its desirable performance.

Using Basic Cassandra SQL Commands

- Create Table

```
CREATE TABLE User(
  id uuid PRIMARY KEY,
  username varchar,
  password varchar,
  displayname varchar,
  email varchar,
);
```

- Read Data
  - SELECT * FROM student;
- Insert Data

```
insert into User( id, username, password, displayName, email )
values( uuid(), 'jack', 'secret', 'Jack', 'jack@example.com' );
```

- Update Data
  - UPDATE student SET city='San Fransisco’ WHERE en=002;
- Delete Data
  - DELETE phone FROM student WHERE en=003;

Use of Cassandra in real scenario

eBay

- involve itself with large data centers, quick solution, large storage and play with unstructured/structured data at runtime.
- Every individual has its own priority, so that handling of such scenario eBay start a recommendation system that work on personalize basis with quick analysis.

Hulu

- relies completely on Cassandra to streaming videos without interruption.
- Problem which Hulu faced;
  - usages of multiple devices, real time accessing and no interruption.
  - Volume of data and its real time solution always supposed to have this application.
  - Scaling of writes from millions of users and in parallel read operation were the major challenges.
- Hbase uses HDFS can lead to single point of failure
- Cassandra also performs ahead with its performance with availability and replication (CAP)

Future of NoSQL DBs

- Cockroach DB
  - promises support for distributed transaction in a geo-replicated way with read/write scalability.
  - millions of users and in parallel read operation were the major challenges.
  - https://github.com/cockroachdb/cockroach
- Foundation DB
  - support for multiple types of data models (key/value, document model, SQL type table, etc.) and the support for properties like ACID

CAP Theorem

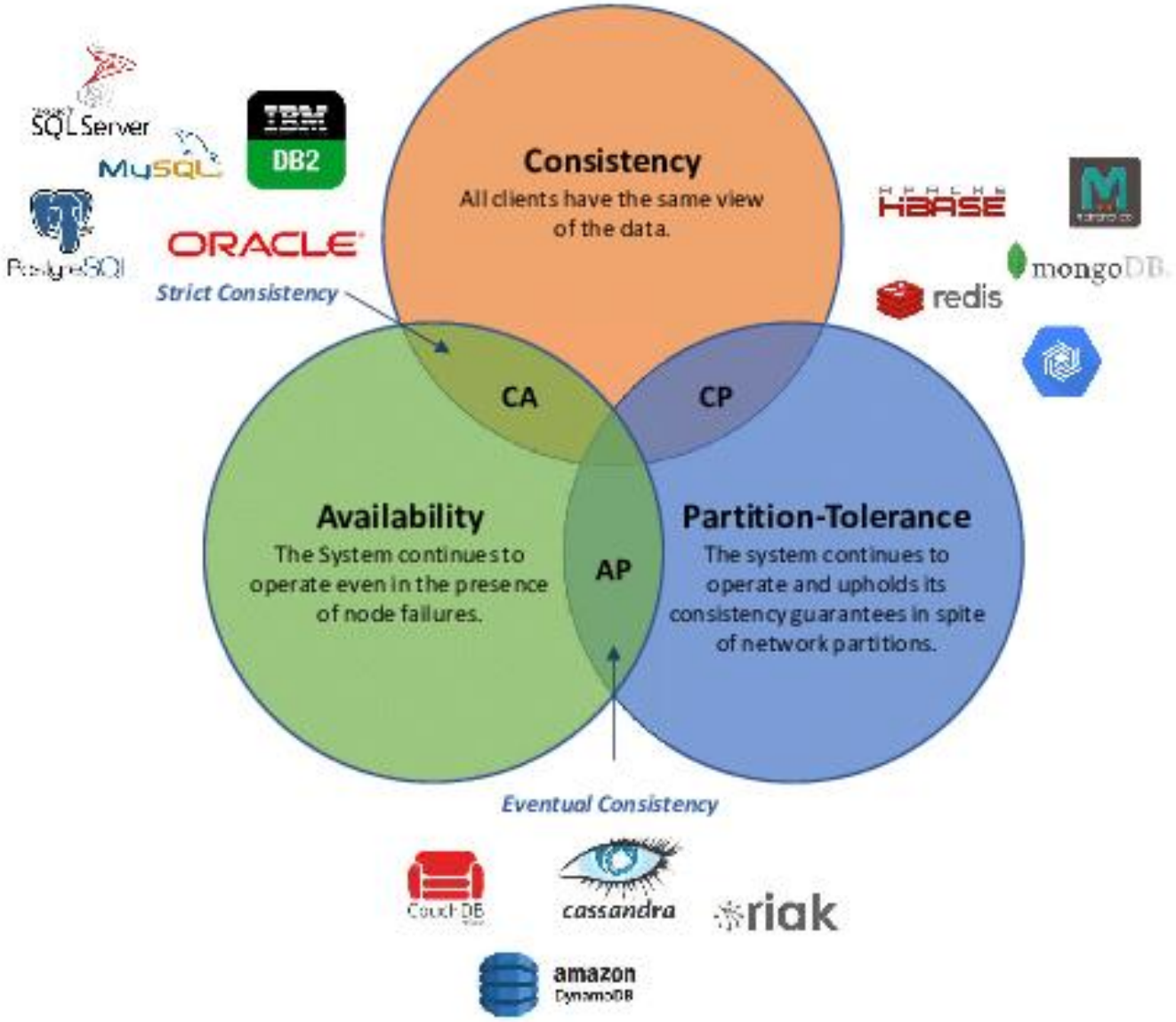
- NoSQL database relaxing with consistency and allow operations to perform in parallel.
- ***CAP theorem*** that defines stages of **consistency**, **availability** and **partition tolerance**.

Consistency	Every read receives the most recent write causing all nodes to return the same data or an error
Availability	Every request receives a response
Partition Tolerance	The system continues to operate despite an arbitrary number of messages being dropped or delayed by the network between nodes

CAP Theorem

- a database cannot exhibit more than two of the three properties at the same time of the CAP theorem.
- **Consistency and availability** (CA)—If the system requires consistency (C) and availability (A), then the available nodes have to communicate to guarantee consistency (C) in the system; hence, network partitioning is not possible.
- **Consistency and partition tolerance** (CP)—If the system requires consistency (C) and partition tolerance (P), availability of the system is affected while consistency is being achieved.
- **Availability and partition tolerance** (AP)—If the system requires availability (A) and partition tolerance (P), consistency (C) of the system is forfeited as the communication between the nodes is broken so the data will be available but with inconsistency.

CAP Theorem


Consistency
All clients have the same view of the data.
Availability
The System continues to operate even in the presence of node failures.
Partition-Tolerance
The system continues to operate and upholds its consistency guarantees in spite of network partitions.
CA
CP
AP
Strict Consistency
Eventual Consistency
SQL Server
MySQL
DB2
ORACLE
redis
mongoDB.
cassandra
riak
amazon
DynamoDB