UNIT-VI
Key-value Stores and Semi-structured Data, using JSON and MongoDB

- **Why NoSQL?**
- Relational databases have been the default choice for centralized data storage, especially in the world of enterprise applications your only choice can be which relational database to use.
- Importance of RDBMS
    - ACID properties
    - more flexibility than a file system in storing large amounts of data
    - Multiple teams can access and update data- Concurrency using Transactions
    - Standardisation of DB storage and access models i.e. relations

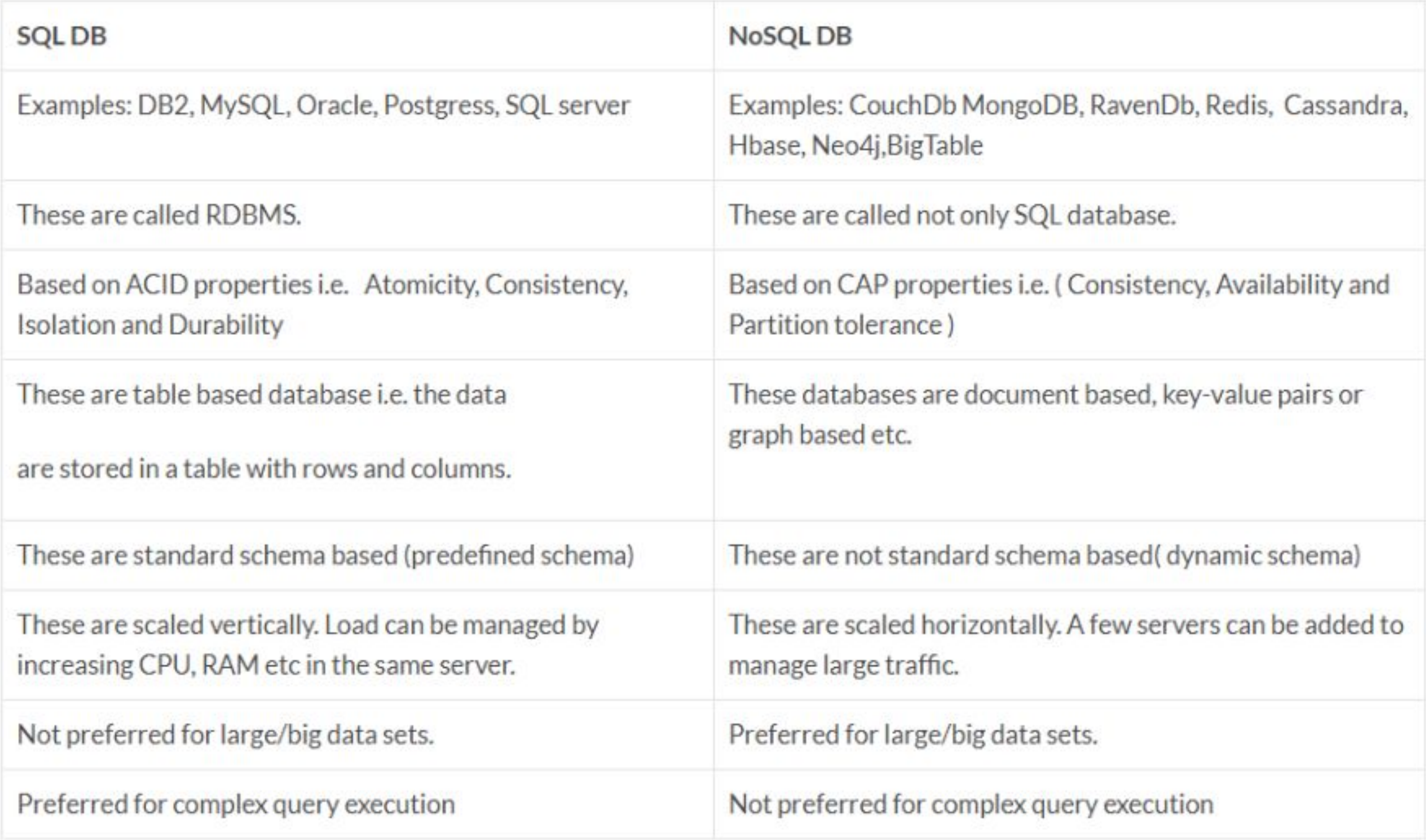
- The Era of Web 2.0
- Introduction of Object oriented DB and object oriented programming languages.

- **Problems with RDBMS**
- Rigid relational structure that’s designed to integrate many applications is more complex than any single application needs.
- If an application wants to make changes to its data storage, it needs to coordinate with all the other applications using the database.
- Different applications have different structural and performance needs,
- In SQL, the data must be structured as relations. However, with a service, you are able to use richer data structures with nested records and lists.
- These are usually represented as documents in XML or, more recently, JSON.

- In 2000s several large web properties dramatically increased in scale.
- Started tracking activity and structure in a very detailed way.
- Large sets of data appeared: links, social networks, activity in logs, mapping data.
- Scaling up implies: bigger machines; more processors; more disk storage; more memory
- But more data also -> internal data fragmentation required in form of clusters
- Issues:
  - Relational databases are not designed to be run on clusters.
  - Clustered relational databases, such as the Oracle Microsoft SQL Server, work on the concept of a shared disk subsystem where cluster still has the disk subsystem as a single point of failure.

- This mismatch between relational databases and clusters led some organization to consider an alternative route to data storage.
- Two companies in particular 1. Google 2. Amazon
- Both were running large clusters
- They were capturing huge amounts of data
- As the 2000s drew on, both companies produced brief but highly influential papers about their efforts:
- BigTable from Google & Dynamo from Amazon
- Thus emerged NoSQL (Not only SQL).
- NoSQL databases operate without a schema, allowing you to freely add fields to database records without having to define any changes in structure first.
- This is particularly useful when dealing with non-uniform data and custom fields

- RDMS is not obsolete but now it offers one option for data storage.
- This point of view is often referred to as polyglot persistence—using different data stores in different circumstances.
- Advantages of NoSQL:
- High Scalability- NoSQL databases use sharding for horizontal scaling.
- High Availability-Auto replication feature in NoSQL databases makes it highly available.
- When should NoSQL be used
- Data is huge
- Schemas, relation and structure are less important

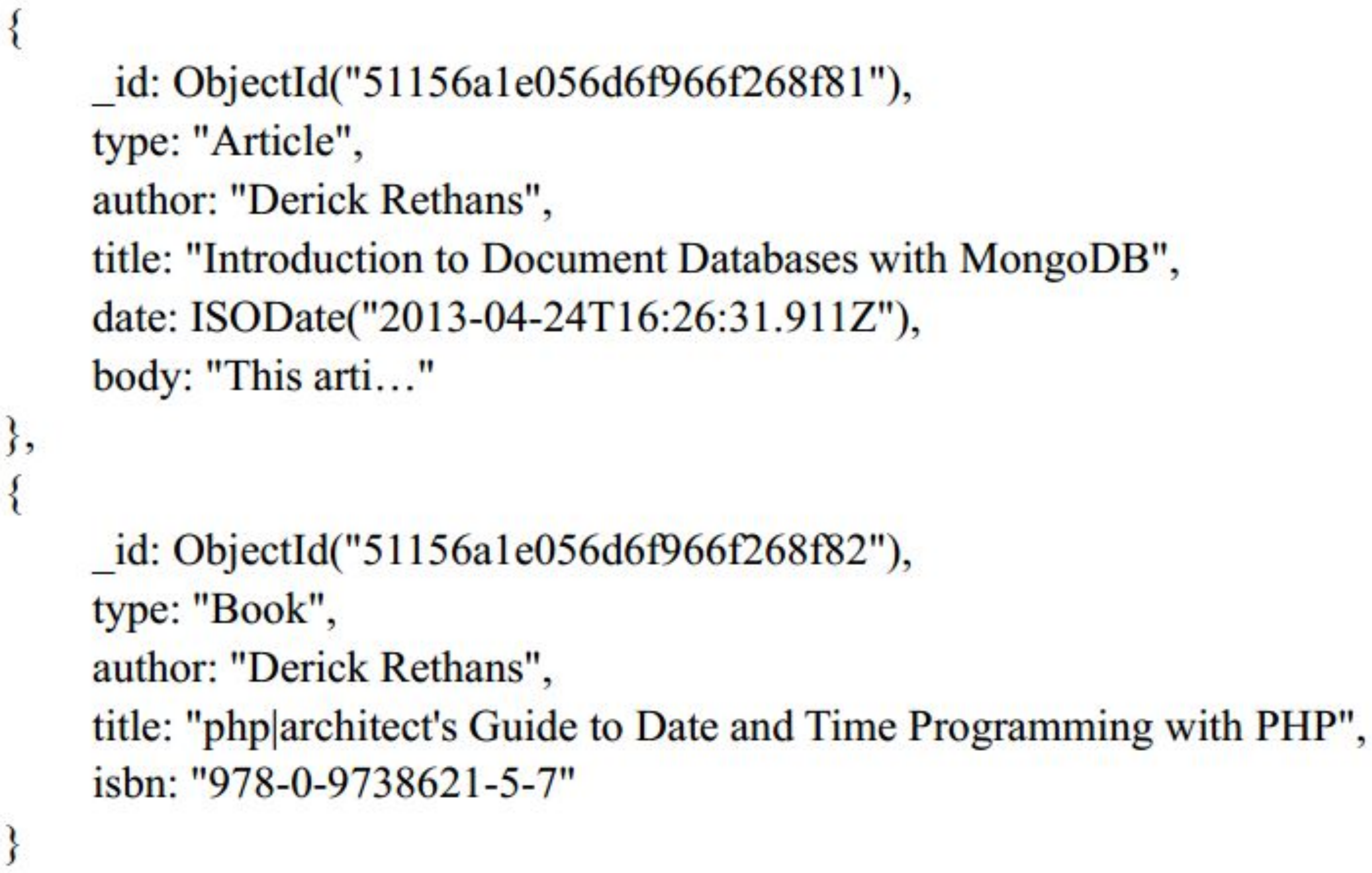


SQL DB	NoSQL DB
Examples: DB2, MySQL, Oracle, Postgress, SQL server	Examples: CouchDb MongoDB, RavenDb, Redis, Cassandra, Hbase, Neo4j,BigTable
These are called RDBMS.	These are called not only SQL database.
Based on ACID properties i.e. Atomicity, Consistency, Isolation and Durability	Based on CAP properties i.e. (Consistency, Availability and Partition tolerance)
These are table based database i.e. the data are stored in a table with rows and columns.	These databases are document based, key-value pairs or graph based etc.
These are standard schema based (predefined schema)	These are not standard schema based(dynamic schema)
These are scaled vertically. Load can be managed by increasing CPU, RAM etc in the same server.	These are scaled horizontally. A few servers can be added to manage large traffic.
Not preferred for large/big data sets.	Preferred for large/big data sets.
Preferred for complex query execution	Not preferred for complex query execution

- NoSQL databases are generally classified into four main categories:
- **Document databases:** These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
- **Key-value stores:** These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
- **Column-family stores:** These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.
- **Graph databases:** These databases store data as nodes and edges, and are designed to handle complex relationships between data.

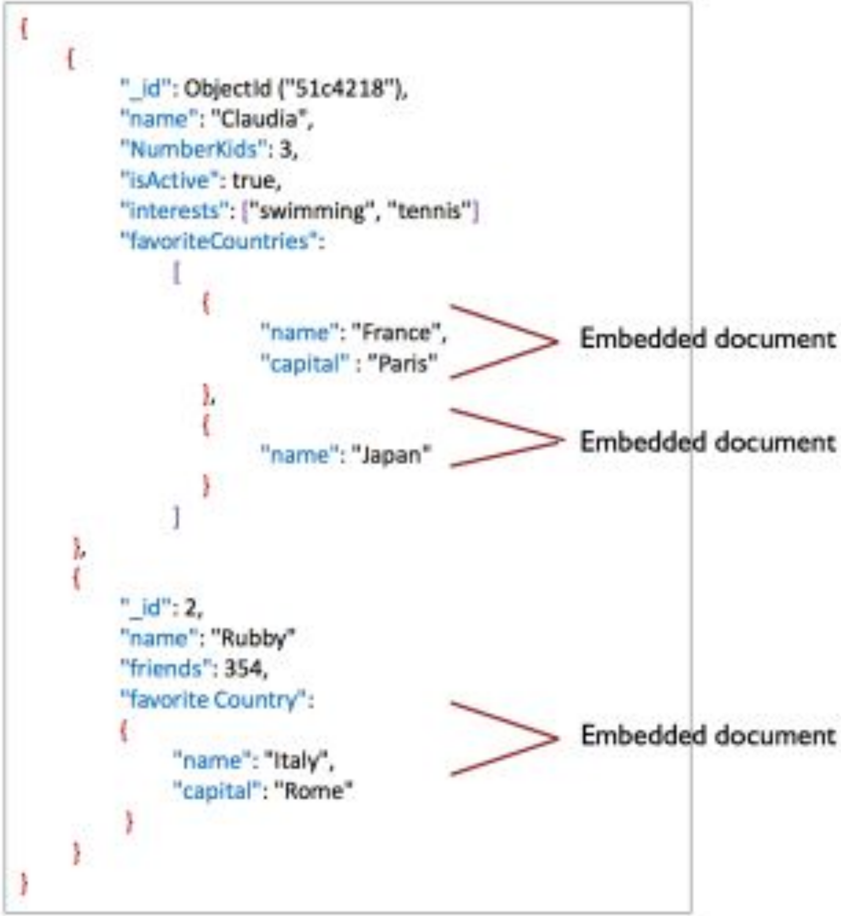
- **Document Databases**
- Loosely structured sets of key/value pairs in documents, e.g., XML, JSON (JavaScript Object Notation)
- Encapsulate and encode data in some standard formats or encodings
- Are addressed in the database via a unique key
- Documents are treated as a whole, avoiding splitting a document into its constituent name/value pairs
- Allow documents retrieving by keys or contents
  - MongoDB (used in FourSquare, Github, and more)
  - CouchDB (used in Apple, BBC, Canonical, Cern, and more)
- The central concept is the notion of a "document“ which corresponds to a row in RDBMS.
- Documents are schema free, i.e., different documents can have structures and schema that differ from one another. (An RDBMS requires that each row contain the same columns.)

A document comes in some standard formats like JSON (BSON).



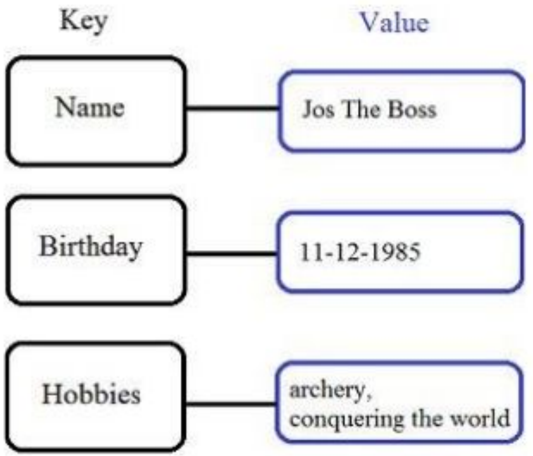
```
{
        _id: ObjectId("51156a1e056d6f966f268f81"),
        type: "Article",
        author: "Derick Rethans",
        title: "Introduction to Document Databases with MongoDB",
        date: ISODate("2013-04-24T16:26:31.911Z"),
        body: "This arti…"
},
{
        _id: ObjectId("51156a1e056d6f966f268f82"),
        type: "Book",
        author: "Derick Rethans",
        title: "php|architect's Guide to Date and Time Programming with PHP",
        isbn: "978-0-9738621-5-7"
}
```

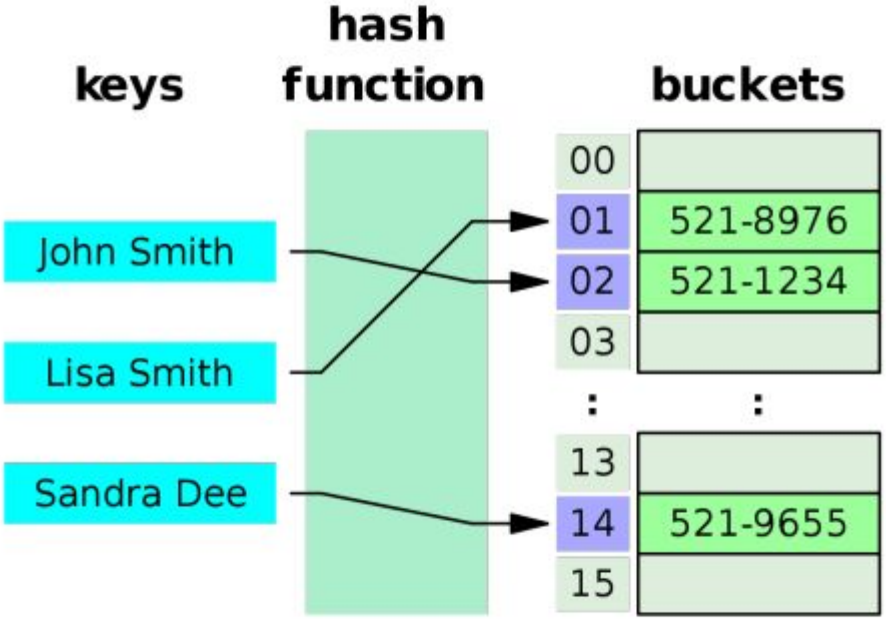
A document can have one or more documents inside.



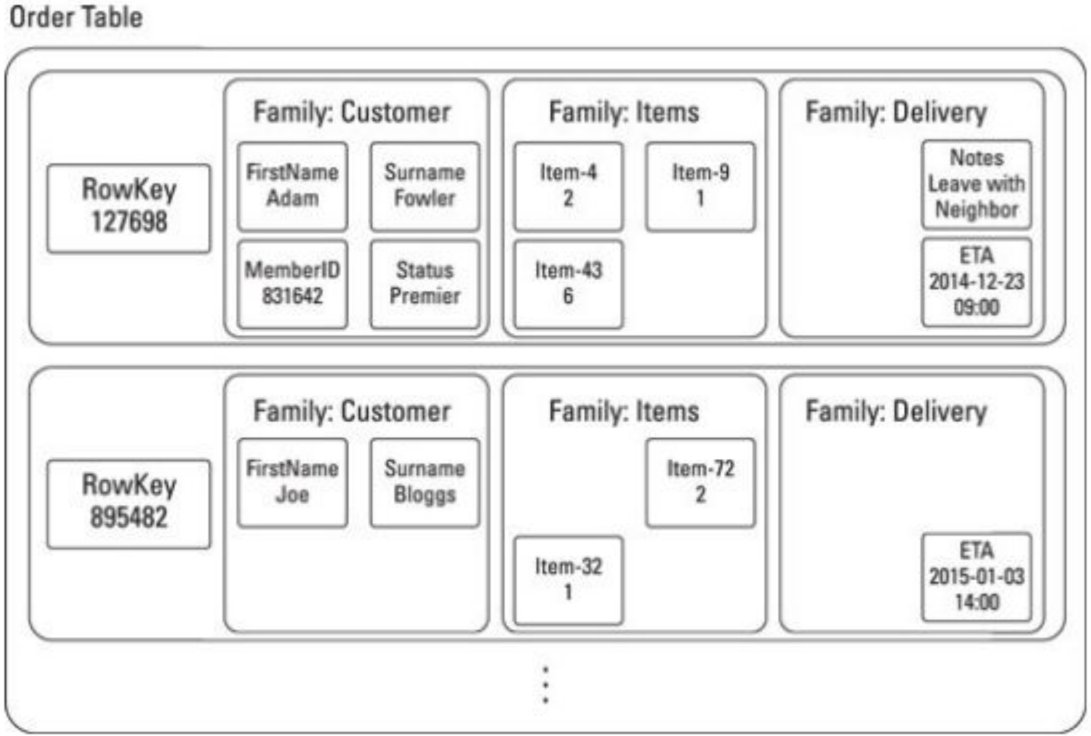
```
{
    {
        "_id": ObjectId ("51c4218"),
        "name": "Claudia",
        "NumberKids": 3,
        "isActive": true,
        "interests": ["swimming", "tennis"]
        "favoriteCountries":
            [
                {
                    "name": "France",
                    "capital" : "Paris"
                },
                {
                    "name": "Japan"
                }
            ]
    },
    {
        "_id": 2,
        "name": "Rubby"
        "friends": 354,
        "favorite Country":
        {
            "name": "Italy",
            "capital": "Rome"
        }
    }
}
```

- **Key value stores**
- Store data in a schema-less way like, stores data as maps, HashMaps or associative arrays
- Provide a very efficient average running time algorithm for accessing data
- Notable for:
  - Couchbase (Zynga, Vimeo, NAVTEQ, ...)
  - Redis (Craiglist, Instagram, StackOverfow, flickr, ...)
  - Amazon Dynamo (Amazon, Elsevier, IMDb,
  - Apache Cassandra (Facebook, Digg, Reddit,
  - Voldemort (LinkedIn, eBay, …)




hash function
keys
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
:
13
14
15
521-8976
521-1234
:
521-9655

- **Column-Family stores**
- Data are stored in a column-oriented way
- Data efficiently stored
- Avoids consuming space for storing nulls
- Columns are grouped in column-families
- Data isn’t stored as a single table but is stored by column families
- Unit of data is a set of key/value pairs
- Identified by “row-key”
- Ordered and sorted based on row-key
- Notable for:
  - Google's Bigtable (used in all Google's services)
  - HBase (Facebook, StumbleUpon, Hulu, Yahoo!, ...)



Order Table

RowKey 127698

Family: Customer
FirstName Adam
Surname Fowler
MemberID 831642
Status Premier

Family: Items
Item-4 2
Item-9 1
Item-43 6

Family: Delivery
Notes Leave with Neighbor
ETA 2014-12-23 09:00

RowKey 895482

Family: Customer
FirstName Joe
Surname Bloggs

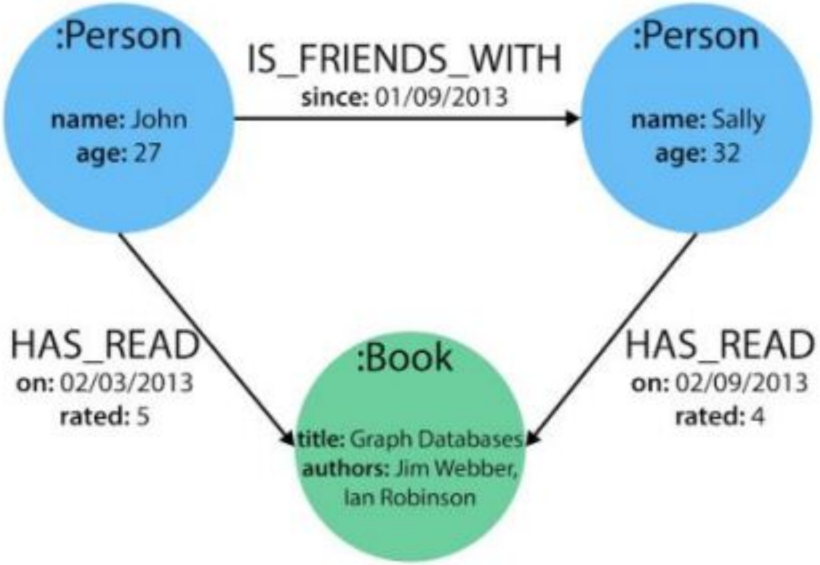
Family: Items
Item-72 2
Item-32 1

Family: Delivery
ETA 2015-01-03 14:00

⋮

Note that a row does not need to have an entry for all columns

- **Graph-oriented Databases**
- Everything is stored as an edge, a node or an attribute.
- Each node and edge can have any number of attributes.
- Both the nodes and edges can be labelled.
- Labels can be used to narrow searches.

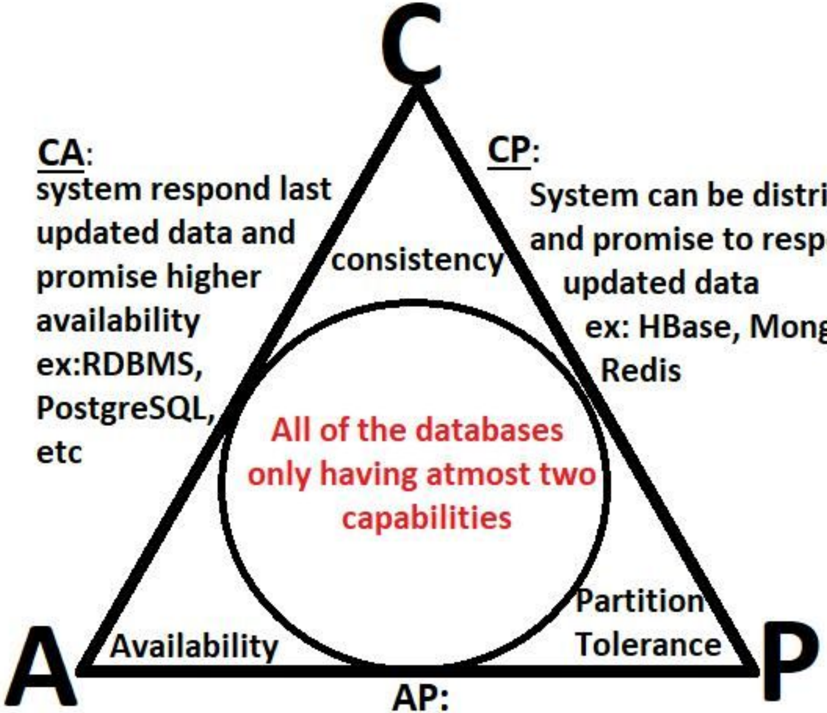


- Issues with scaling up when the dataset is just too big
- RDBMS were not designed for distributed databases
- Traditional DBMSs are best designed to run well on a “single” machine
- Larger volumes of data/operations requires to upgrade the server with faster CPUs or more memory known as ‘scaling up’ or ‘Vertical scaling’
- NoSQL solutions are designed to run on clusters or multi-node database
- Solutions:
- Larger volumes of data/operations requires to add more machines to the cluster, Known as ‘scaling out’ or ‘horizontal scaling’
- Different approaches include:
  - Master-slave
  - Sharding (partitioning)

- **BASE Transactions**
- Acronym contrived to be the opposite of ACID
- **B**asically **A**vailable (Failure will not halt system)
- **S**oft state (system state will change over time)
- **E**ventually Consistent (system will be consistent over time)
- Characteristics:
  - Weak consistency – stale data allowed
  - Availability first
  - Best effort
  - Approximate answers allowed
  - Aggressive (optimistic)
  - Simpler and faster

- **CAP Theorem**
- A congruent and logical way for assessing the problems involved in assuring ACID-like guarantees in distributed systems is provided by the
- CAP theorem
- At most two of the following three can be maximized at one time
  - Consistency-Each client has the same view of the data
  - Availability-Each client can always read and write
  - Partition tolerance-System works well across distributed physical networks
  - Eventual Consistency using Gossip Protocol (ex-graphs)

- Consistency and Availability is not “binary” decision
  - AP systems relax consistency in favor of availability – but are not inconsistent
  - CP systems sacrifice availability for consistency- but are not unavailable
  - This suggests both AP and CP systems can offer a degree of consistency, and availability, as well as partition tolerance



CA:
system respond last
updated data and
promise higher
availability
ex:RDBMS,
PostgreSQL,
etc

CP:
System can be distributed
and promise to respond last
updated data
ex: HBase, MongoDB,
Redis

All of the databases
only having atmost two
capabilities

AP:
System can be ditributed and
promise to has high availability
ex: Cassandra, CouchDB, Riak,
Voldemort, DynamoDB, etc