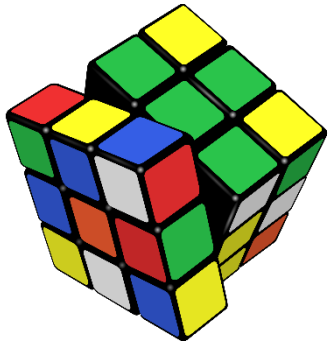


Big Data Engineering For Analytics

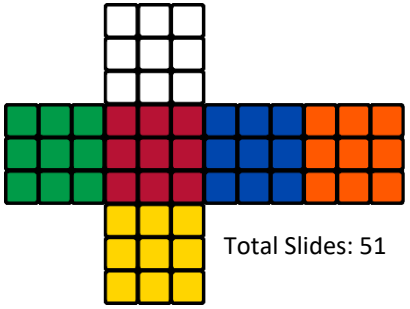
Big Data Storage Patterns

Polyglot storage & NoSQL Focus

Dr Venkat Ramanathan

(rvenkat@nus.edu.sg)

NUS-ISS



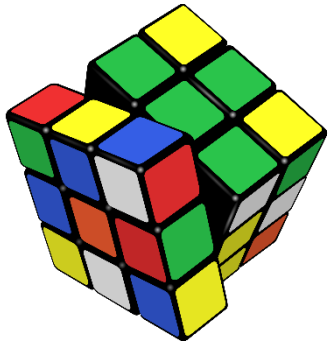
Total Slides: 51

Learning Objectives

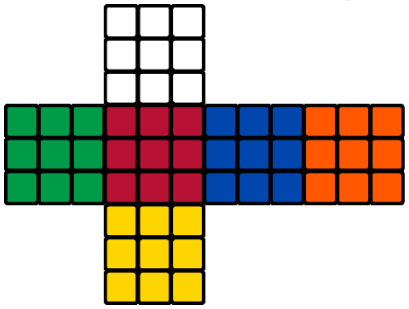
- Understand how data storage is impacted in Big Data scenario.
- Understand the challenges posed by Big Data evolutions and the various data engineering technologies to address these challenges.
- Describe the data aggregation and their four main manifestation as NoSQL data models.
- Identify how relevant or different are relational databases to NoSQL database storage.
- Analyze how NoSQL databases distribute data to run on cluster by sharding and replication.
- Judge when to utilize NoSQL technologies for maximum benefit based on CAP theorem and quorums.

Agenda

- Why NoSQL? Motivations behind NoSQL
- Characteristics of NoSQL
- Aggregated Data Models and Types.
  - Key-Value Databases
  - Column Databases
  - Document Databases
  - Graph Databases
- CAP Theorem and Sharding
- Popular Examples
- Summary



Big Data Engineering For Analytics

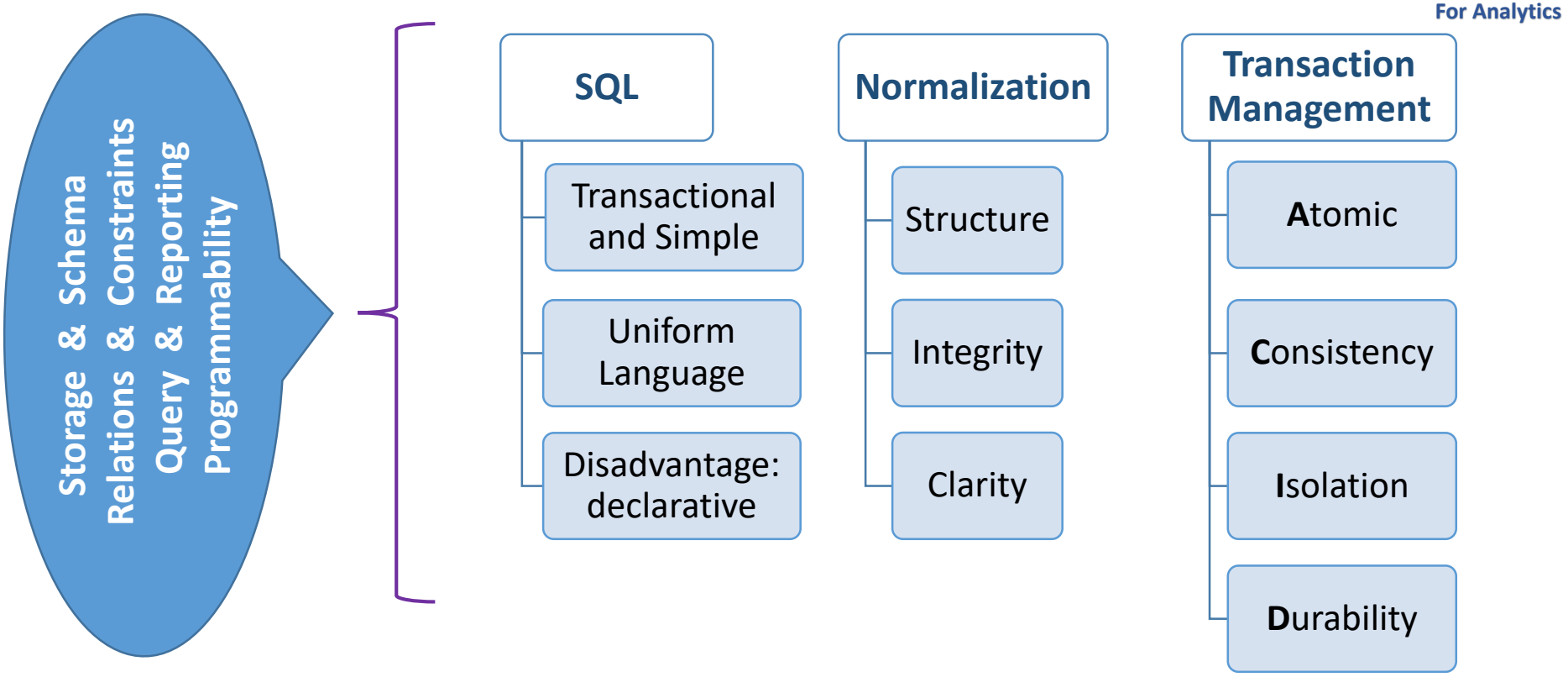


Data Storage – Past, Present and the Future

History & Types of Storing Data

- File Based Storage (early days, 1960s onward)
- Schemas – the need and purpose
- Hierarchical Model
- Network Model
- Relational Model
- Object Model

In Summary, RDBMS Features …

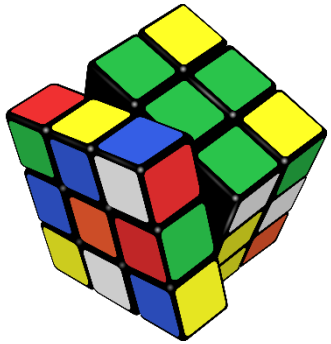


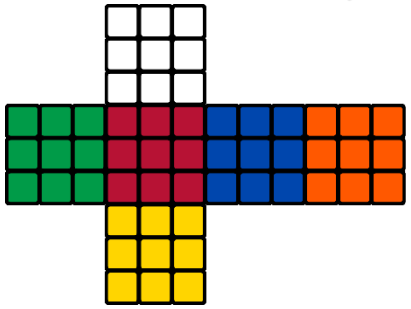
And In Effect Offers…

Mature Technology → Precise Querying → Star Schema

But is…

Expensive when scaling





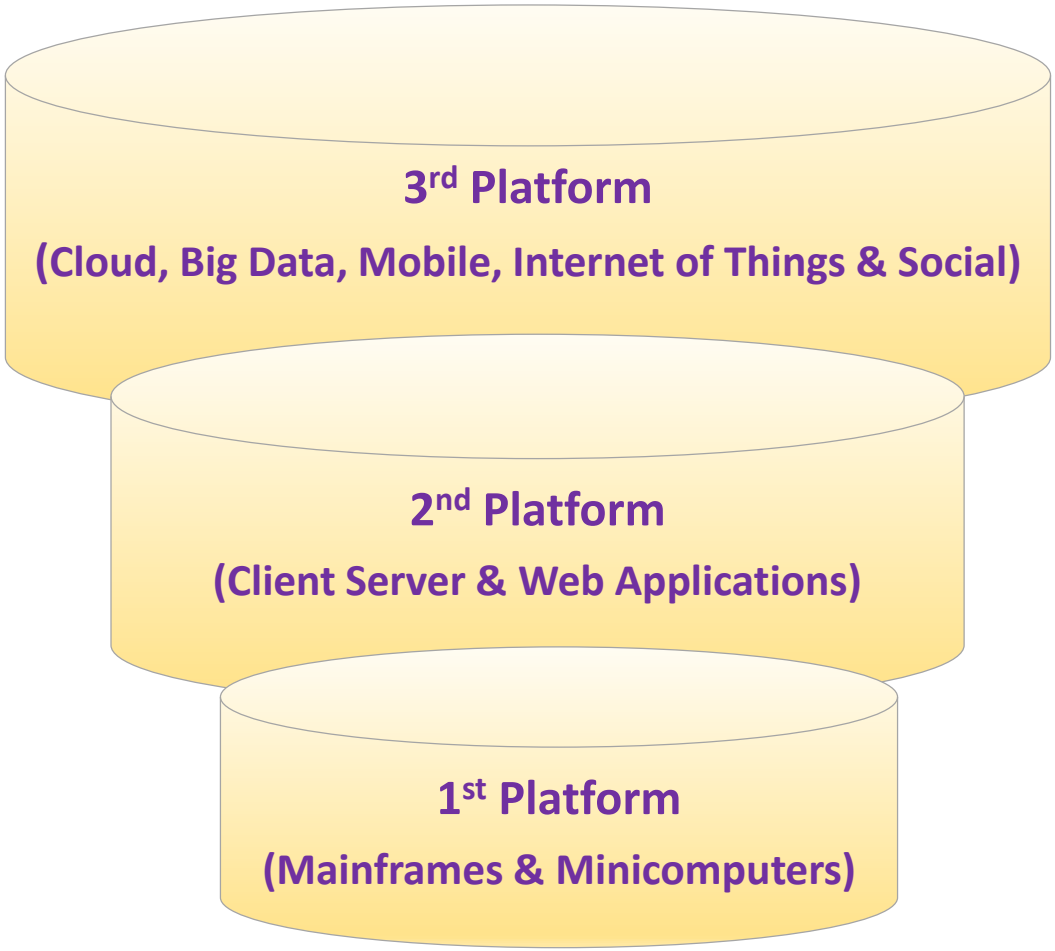
Why NoSQL? Motivations behind NoSQL

"Study after study shows that the very best designers produce structures that are faster, smaller, simpler, clearer, and produced with less effort. The differences between the great and the average approach an order of magnitude.

Fred Brooks

Evolution of Systems: Computing Paradigm

IDC's "Three accommodating platforms"



A byte is a unit of digital information that consists of 8 bits. In the International System of Units (SI) scheme every 1,000 (10^3) multiple of a byte is given a distinct name, which is as follows:

- *Kilobyte (kB) — 10^3*
- *Megabyte (MB) — 10^6*
- *Gigabyte (GB) — 10^9*
- *Terabyte (TB) — 10^{12}*
- *Petabyte (PB) — 10^{15}*
- *Exabyte (EB) — 10^{18}*
- *Zettabyte (ZB) — 10^{21}*
- *Yottabyte (YB) — 10^{24}*

Reference: http://www.idc.com/prodserv/3rd-platform/

*International **Data** Corporation*

Hence, what is happening to DATA of late?

- Effectively BIG DATA now.
- Recall:
  - **Volume** refers to the vast amount of data generated every second.
  - **Variety** refers to the different types of data we can now use.
  - **Velocity** refers to the speed at which new data is generated and the speed at which data moves around.
  - **Veracity** refers to the uncertainty or trustworthiness of the data.
  - **Variability** refers to the change in data meaning/models over time.
  - **Value** refers to our ability turn our data into value.

- Can the RDBMS (aka loosely called SQL DB) handle the above?
  - No (say the gurus and industry).

Motivations for Alternate Storage Models?

- Alternates are needed:
  - Guys who got stuck viz., Google, Amazon started doing their own Databases…
- *Contributing factors were mainly the limitations of RDBMS …*
  - *Variety: the need for handling Non-static Schema*
    - Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
  - *Variability – the need to accept Heterogeneity*
    - Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.
  - *Volume of Data – the need for Scalability*
    - Organizations are now turning to scale-out architectures using open source software, commodity servers & cloud computing instead of large monolithic DB servers.
  - *Velocity – the need for Agility*
    - Now a days small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.

Evolution of Non-RDBMBS storage systems

- The following organizations faced Big Data situations in the 2000s and chose to evolve alternate storage patterns listed in brackets:

2004 Google (BigTable)

2007 Amazon (Dynamo)

2008 Facebook (Cassandra)

Yahoo (Hadoop/HBase)

LinkedIn (Project Voldemort)

Google's Requirement

Question	Response
What was their focus?	Search Engine *(i.e, in 2003 when the requirement came)*
What they needed from Database?	Input: Key word Output: List of web pages
What was their challenge?	My God, the web is exploding, pages are increasing, users are demanding, etc… I need to store lots of records. I don't have too many diverse entities and not many fields in entities. I need to retrieve data fast and display to users. But, hey, my users do not go edit data.
Why not live with RDBMS?	I don't need transactions, I don't need multiple entities, I don't need integrity constraints, etc. etc. So do you think I am getting a product that offers too much? Ok but I have to put data in multiple servers due to volume and response time needs – RDMS suks…

Google's Decision

- Discard RDBMS
- Create my own
  - Sigh,
    - who in business is talking of not wasting resources re-inventing wheels now?
    - who in IT is not advocating reusability mantra of software engineering etc?
- Ok what model do I use:
  - Key -> Value (stated already)
  - Key – is a list of key words
  - Value – is a list of web page urls

  - Ok lets get some Computer Science into it (not in detail though)
    - We may need in this: Only one Index, Sorted Lexicographically on row key, etc…

Birth of NoSQL

- Carlo Strozzi (1998)
  - Named NoSQL for small isolated non-relational databases
- Johan Oskarsson (2009)
  - Used this term as # tag for a conference to discuss the mushrooming databases from Google, Amazon and other players.
- We live with this
  - But strong view exists it is typically not Non-SQL, but Not Only SQL
  - Essentially a family of various types of databases.

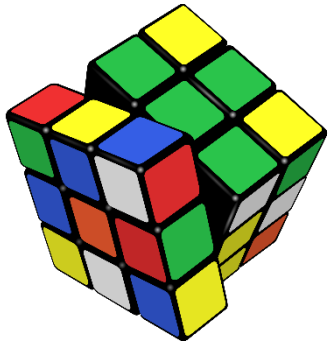
- Two Major Trends that influenced NoSQL:
  - *Exponential **growth** of data generated by new systems and users*
  - *Increasing **interdependency** and complexity of data*

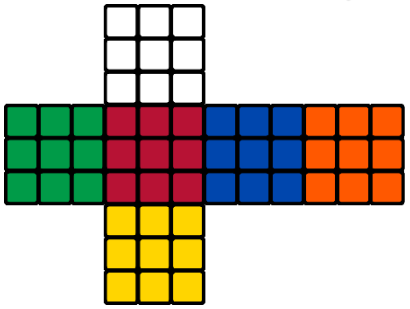
How does NoSQL address the motivations?

- **NoSQL accommodates schema-less data storage,** which is one of the main advantages.
  - It will allow the storing of all types of data in different formats and in different schemas, thereby providing more robust and agile development.
- **NoSQL driven platform accommodates servers to scale horizontally**, which means it is very easy to scale the capacity up or down.
  - We can simply add new servers or remove servers to increase or decrease its capacity, storage, and computation power.
- **NoSQL structure facilitates working in a clustered environment** which is much less expensive than a highly reliable server.
  - Server clusters are mainly built on commodity hardware, without affecting the performance or reliability.
- **NoSQL databases spread across multiple nodes** which eases replication to multiple servers.
  - Some of NoSQL database frameworks even work without the master slave concept, which makes them highly available with no single point of failure.

Tangible benefits of NoSQL

- *Continuously available*—This is available all the time even in the face of the most devastating infrastructure outages.
- *Geographically distributed*—Database instances are distributed and available everywhere you need it.
- *Operationally low latency*—The response time is very less and is suitable for intense operational applications.
- *Linearly scalable*—Database servers could be added in order to tackle higher user as well as date nodes. The system performance grows linearly with the addition of new servers in the cluster.
- *Lower total cost of ownership*—The leverage of commodity servers and open-source software for NoSQL database management systems results in lower TCO and higher ROI.
- *Multiple types*—There are specific NoSQL types for specific application requirements and data models.





Data Types

Data can take different forms.

- ***Structured data*** refers to data that has a defined length and format
  - Computer- or machine-generated: Machine-generated data generally refers to data that is created by a machine without human intervention.
  - Human-generated: This is data that humans, in interaction with computers, supply.
- ***Unstructured data*** is data that does not follow a specified format
- ***Semi-structured data*** refers to data that falls between structured and unstructured data.
  - Semi-structured data does not necessarily conform to a fixed schema (that is, structure) but can be interpreted (mainly by self-describing data like having simple label/value pairs).

Structured *vs* Unstructured Data *vs* Semi-structured

- Structured Data
  - Information with a high degree of organization.
  - The structure facilitates data processing.
  - Typically has a data model.
  - Easily entered, stored, queried, analyse.
  - The storage understand the structure.

List of students
Employee details

- Unstructured Data
  - Data that does not have a schema.
  - Cannot be queried or processed in typical sense
  - For inference may require NLP, Image Processing, ML etc

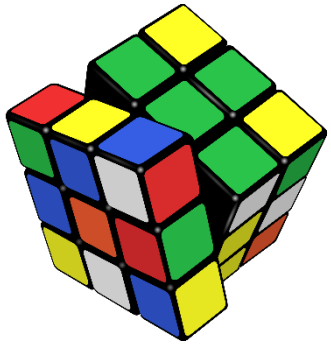
PDF files
Audio
Video

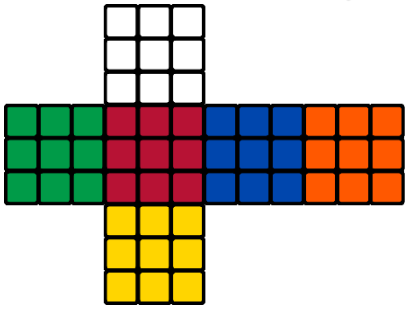
- Semi Structured
  - Has a schema but is not rigid.
  - Easy to adapt to changes
  - Easy to split the dataset and distribute

Employee data stored in XML JSon

Examples of unstructured data

- Examples of Machine-generated Unstructured Data:
  - Satellite images: includes weather data or the data that the government captures in its satellite surveillance imagery.
  - Scientific data: includes seismic imagery, atmospheric data, and high energy physics.
  - Photographs and video: includes security, surveillance, and traffic video.
  - Radar or sonar data: includes vehicular, meteorological, and oceanographic seismic profiles.
- Examples of Human-generated Unstructured Data:
  - Text internal to your company: includes all the text within documents, enterprise information, logs, survey results, and e-mails.
  - Social media data: includes data from the social media platforms such as YouTube, Facebook, Twitter, LinkedIn, and Flickr.
  - Mobile data: includes text messages and location information.
  - Website content: includes data from any site delivering unstructured content, like YouTube, Flickr, or Instagram.





Polyglot Persistence

The Era of Polyglot Persistence

- Polyglot persistence is about using different data storage technologies to handle varying data storage needs
- Polyglot persistence can apply across an enterprise or within a single application
- Encapsulating data access into services reduces the impact of data storage choices on other parts of a system
- Adding more data storage technologies increases complexity in programming and operations, so the advantages of a good data storage fit need to be weighed against this complexity

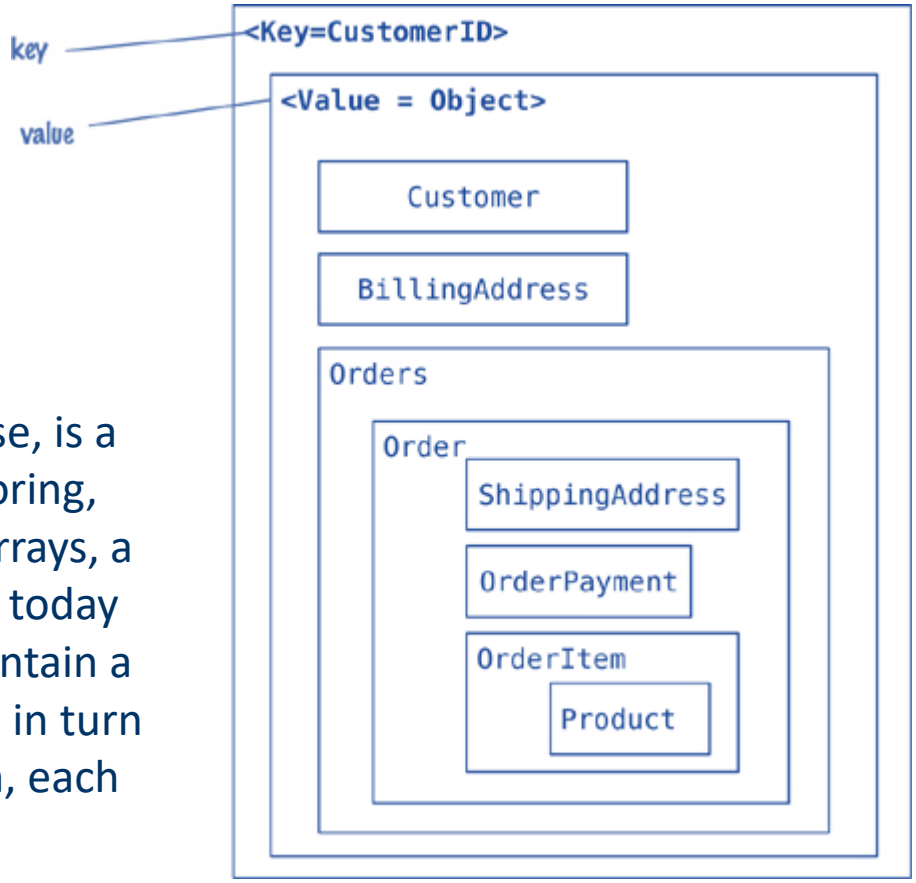
Polyglot Persistence is a fancy term to mean that when storing data, it is best to use multiple data storage technologies, chosen based upon the way data is being used by individual applications or components of a single application.

NoSQL Data Architecture Patterns

Pattern name	Description	Typical uses
Key-value store	A simple way to associate a large data file with a simple text string	Dictionary, image store, document/file store, query cache, lookup tables
Column family store	A way to store sparse matrix data using a row and a column as the key	Web crawling, large sparsely populated tables, highly-adaptable systems, systems that have high variance
Document store	A way to store tree-structured hierarchical information in a single unit	Any data that has a natural container structure including office documents, sales orders, invoices, product descriptions, forms, and web pages; popular in publishing, document exchange, and document search
Graph store	A way to store nodes and arcs of a graph	Social network queries, friend-of-friends queries, inference, rules system, and pattern matching

Key Value Store

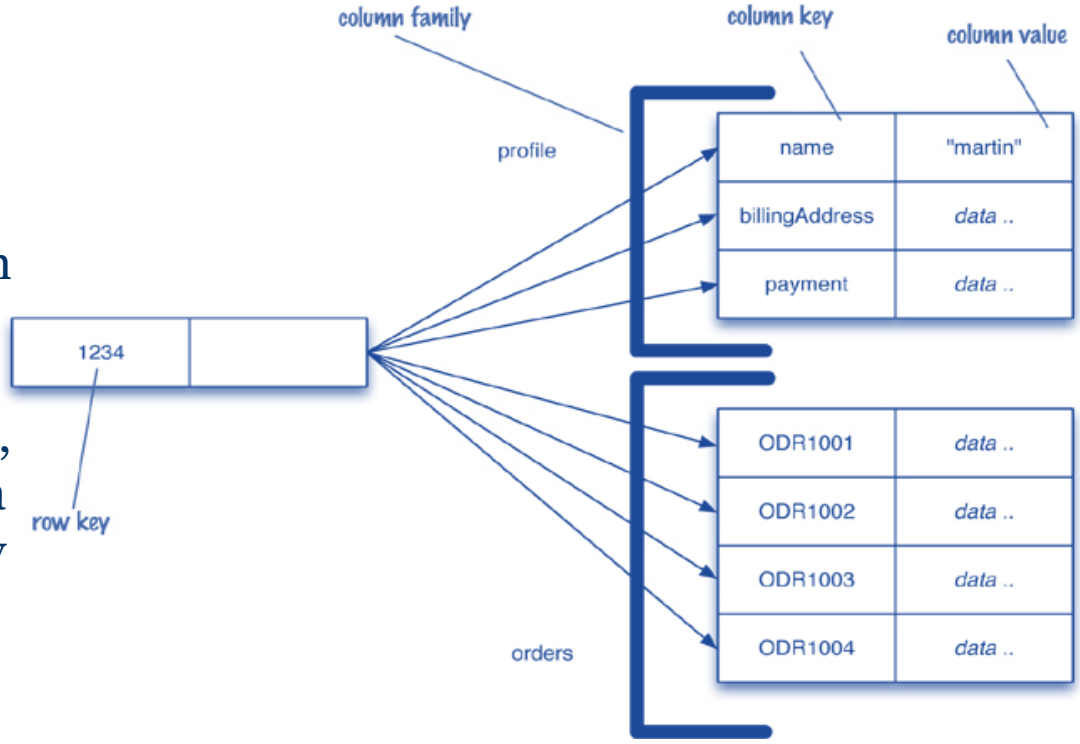
Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623



A key-value store, or key-value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data.

Column Family Data Store

Column-family databases organize their columns into column families. Each column has to be part of a single column family, and the column acts as unit for access, with the assumption that data for a particular column family will be usually accessed together.

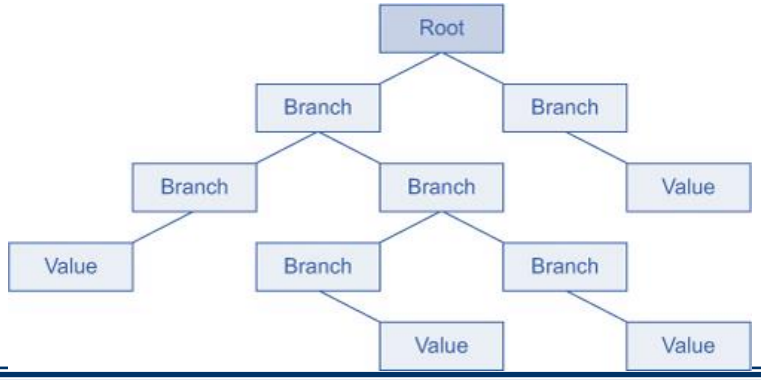


Row-oriented: Each row is an aggregate (for example, customer with the ID of 1234) with column families representing useful chunks of data (profile, order history) within that aggregate.
Column-oriented: Each column family defines a record type (e.g., customer profiles) with rows for each of the records. You then think of a row as the join of records in all column families.

Document Store

In a document data store everything related to a database object is encapsulated together. Thus:

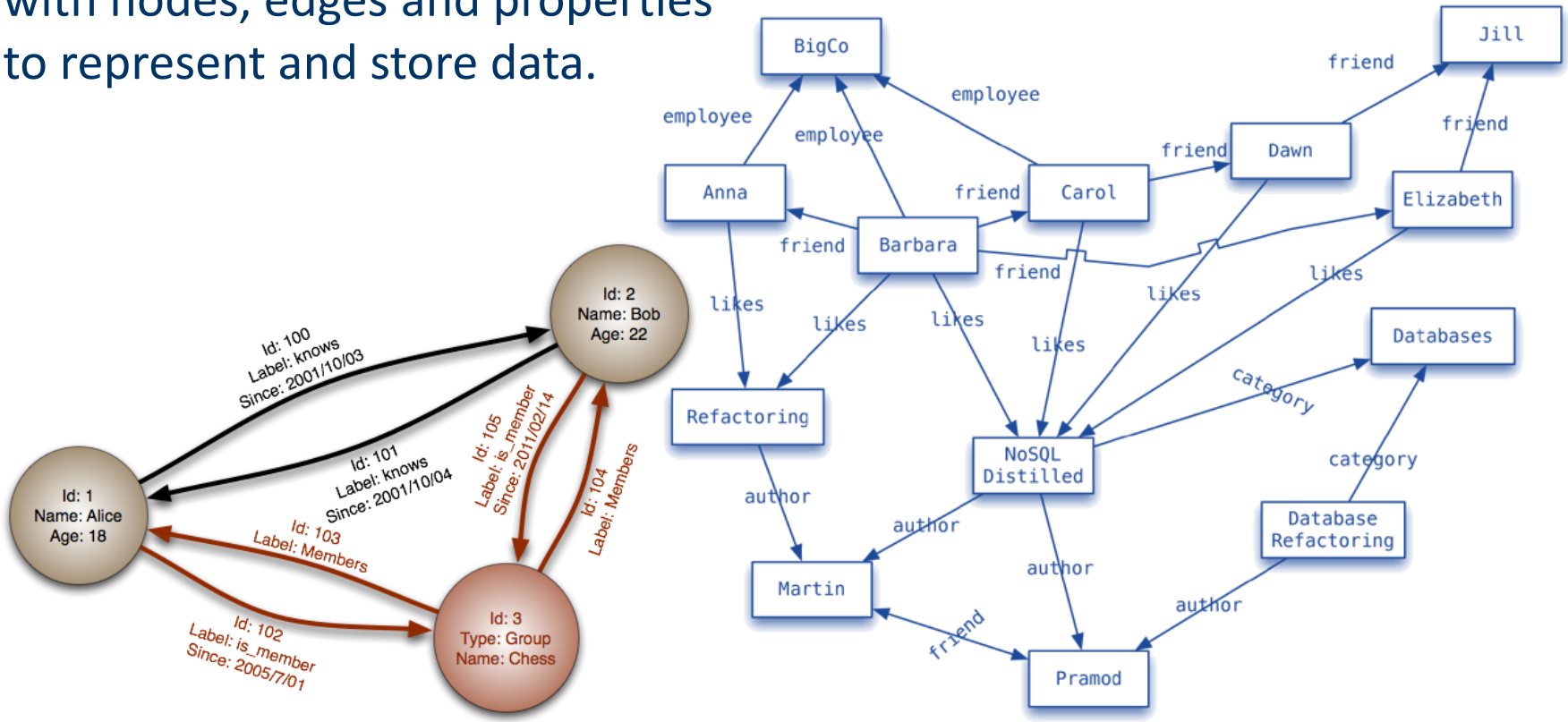
- **Documents are independent units** with better performance and easy distribution of data across multiple servers.
- **Application logic is easier to write**. We can turn the object model directly into a document.
- **Unstructured data can be stored easily** and additionally the information schema structure is known in advance.

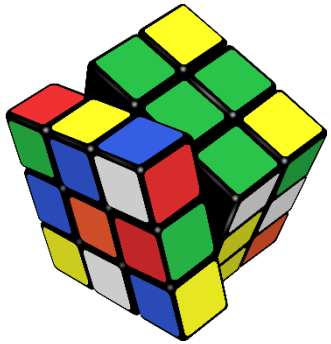


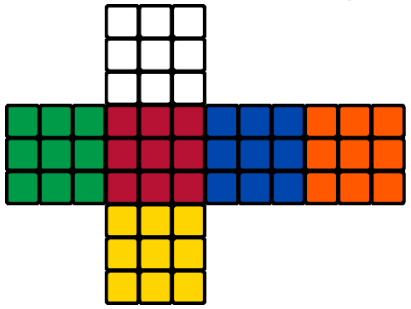

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

Graph Data Store

Graph data store uses graph structures for semantic queries with nodes, edges and properties to represent and store data.







Choosing Data Store & Available Products

When to choose KV? Eg: Redis

- User-defined indexing schemes.
- Message queues with real-time new element notification.
- Convert to directed and undirected graph stores for following or friending systems.
- Real-time publish/subscribe notification systems.
- Real-time analytics back ends.
- Bloom filters servers.
- Task queues and job systems.
- High score leaderboards.
- User ranking systems.
- Hierarchical/tree structured storage systems.
- Individual personalized news or data feeds for your users.

Key Value Types and Notable Examples

KV - eventually consistent models

- Apache Cassandra [* popular]
- Dynamo
- Oracle NoSQL Database
- Project Voldemort
- Riak [* popular]
- OpenLink Virtuoso

KV – serially ordered consistency models

- Berkeley DB
- FairCom c-treeACE/c-treeRTG
- FoundationDB
- HyperDex
- IBM Informix C-ISAM
- InfinityDB
- LMDB
- MemcacheDB
- NDBM

KV – random consistency models

- Aerospike
- Coherence
- FairCom c-treeACE
- Hazelcast
- memcached
- OpenLink Virtuoso
- Redis
- XAP
- Gemfire

When to choose Column Family
Eg: Cassandra, DynamoDB

- There exists relatively clear structure and static nature of the schema
- When data is large
  - Need to distribute across servers (scalability)
  - Need to perform compression
- When the data attributes have a small group of collective data that are more often queried.
- When there is significant aggregation type queries such as in Analytics

When to use document store? Eg: Mongo

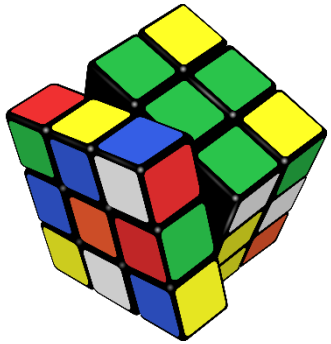
- ***Data insert consistency***: If there is a requirement of writing huge amount of data without losing some data, then MongoDB is the best suited because of its high data insertion rate.
- ***Data corruption recovery***: In MongoDB, data repair can be done on a database level and there is an automatic command to do so. However, the command reads all the data and rewrites it to a new set of files which may be time consuming if the database is huge but it is preferred over losing the entire dataset.
- ***Load balancing***: MongoDB supports faster replication and automatic load balancing configuration because of data placed in shards.
- ***Avoid joins***: If the developer wants to avoid normalization and joins, MongoDB is the best suited.
- ***Changing schema***: MongoDB is schema-less, hence adding new fields will not result in any issues.
- ***Not relational data***: If the data to be stored need not to be a relational one, then MongoDB should be selected.
- ***Mapping***: If the mapping of application data objects is to be done directly into the document-based storage, MongoDB is the best suited.
- ***Creating database cluster***: If the database is geographically distributed and the user wishes to create cluster and speed up data queries among remotely located databases, MongoDB is suitable.

Notable Document Examples

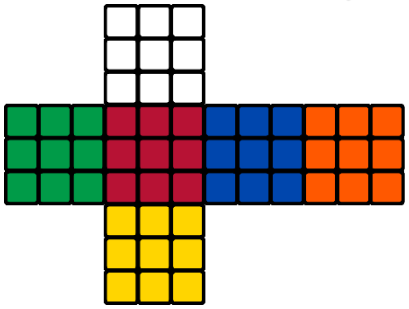
Name	Notes
Cloudant	Distributed database service based on BigCouch, the company's open source fork of the Apache-backed CouchDBproject. Uses JSON model.
Couchbase Server	Distributed NoSQL Document Database, JSON model and SQL based Query Language.
DocumentDB	Platform-as-a-Service offering, part of the Microsoft Azureplatform
Informix	RDBMS with JSON, replication, sharding and ACID compliance
Jackrabbit	Java Content Repository implementation
Lotus Notes (IBM Lotus Domino)	MultiValue
MongoDB	Document database with replication and sharding, BSONstore (binary format JSON)
PostgreSQL	HStore, JSON store
RavenDB	2nd generation document database, JSON format with replication and sharding.
Solr	Search engine
TokuMX	MongoDB with Fractal Tree indexing
OpenLink Virtuoso	Middleware and database engine hybrid

When to choose Graph? Eg: Neo4J

- When entities are to be linked in a networking fashion
  - Social networking such as friends in LinkedIn.
  - Recommender systems such as recommending similar products in ecommerce site.
- Data Relationships
  - Data that exhibits Many-to-Many relationship; *viz., Low Latency at Large Scale*
  - Data that has unstructured relationships *viz., Visualization such as time series and demographics with AI algorithms*
  - Data relationships are prone to change or evolve *viz., intricately structured high value relationships*
- Graphs are useful when navigations between entities require to be fast to improve user experience.
- The size of data is limited in comparison to other data stores.
  - Most graphs are derived from larger data sets in other forms of data stores.



Big Data Engineering For Analytics



Consistency and Scaling

Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious.

Fred Brooks

Sharding & Replication

- There are two styles of distributing data:
  - **Sharding** distributes different data across multiple servers, so each server acts as the single source for a subset of data
  - **Replication** copies data across multiple servers, so each bit of data can be found in multiple places.
  - A system may use either or both techniques.
- Replication comes in two forms:
  - **Master-slave** replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads
  - **Peer-to-peer** replication allows writes to any node; the nodes coordinate to synchronize their copies of the data.
  - Master-slave replication reduces the chance of update conflicts but peer to-peer replication avoids loading all writes onto a single point of failure

Sharding Illustrated

- Horizontal partitioning of data (e.g. hash or range partitioning)
  - **Sharding** puts different data on separate nodes, each of which does its own reads and writes particularly valuable for performance because it can improve both read and write performance
- Example:
  - In an ecommerce shopping system (such as EBay or Lazada), several millions of orders are stored. To improve performance these order records can be distributed (sharded) across multiple server. For instance a geographical sharding can be done by keeping Singapore Customer Orders in *Singapore Server* and Malaysia Customer Orders in *Malaysia Server.*
  - Data Volume gets distributed and customer request access also gets distributed thereby improving both storage efficiency and performance improvement.

What is the advantage of "Data Sharding"?

- Many NoSQL databases offer auto-sharding, where the database takes on the responsibility of allocating data to shards and ensuring that data access goes to the right shard. Consequences:
  - manage parallel access in the application
  - scales well for both reads and writes
  - not transparent, application needs to be partition-aware
- Sharding is valuable for performance since it can improve both read and write performance.
- Using replication, particularly with caching, can greatly improve read performance but does little for applications that have a lot of writes.

Replication Illustrated

- Duplication of Data
  - **Replication** puts same set of data on several nodes.
  - If a Master-Slave replication pattern is used, then all writes are directed to the Master node. Data then gets replicated to the slaves nodes. Slave nodes are used only for read operations. This speeds up read operations, but does little for write performance improvement.
  - If Peer to Peer replication pattern is used, then each server can take part in both read and write operations. This distributes traffic across multiple servers there by both read and write performance gets improved. However it causes challenges for data consistency to a larger extent.
- Example:
  - In an ecommerce shopping system (such as EBay or Lazada), assuming that the Product List is common world-wide then storing at a single server will direct world wide traffic to that server. In such situation it may be prudent to replicate the Product List (aka Table) to several servers so traffic can be distributed to enhance performance.

Types of Consistency

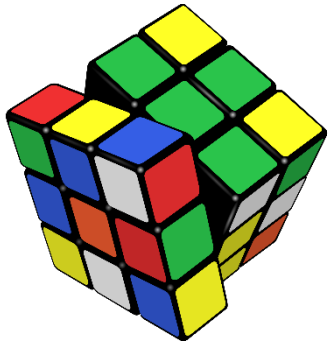
- **Consistency with other users:** If two users query the database at the same time, will they see the same data? Traditional relational systems would generally try to ensure that they do, while non-relational databases often take a more relaxed stance.
- **Consistency within a single session:** Does the data maintain some logical consistency within the context of a single database session? For instance, if we modify a row and then read it again, do we see our own update?
- **Consistency within a single request:** Does an individual request return data that is internally coherent? For instance, when we read all the rows in a relational table, we are generally guaranteed to see the state of the table as it was at a moment in time. Modifications to the table that occurred after we began our query are not included.
- **Consistency with reality:** Does the data correspond with the reality that the database is trying to reflect? For example, it's not enough for a banking transaction to simply be consistent at the end of the transaction; it also has to correctly represent the actual account balances. Consistency at the expense of accuracy is not usually acceptable.

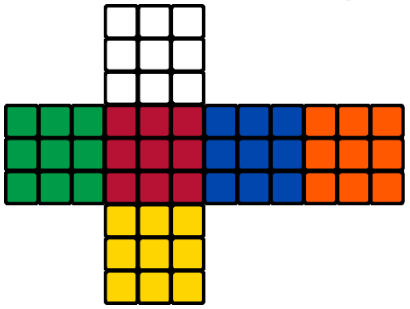
Relaxing ACID properties

- **ACID** is hard to achieve in distributed databases
- not always required
  - E.g. blogs, status updates, product listings, etc. do not need ACID.
  - Big Data Analytics Solutions do not need ACID as it is data retrieval.
- **Availability**
  - Traditionally, thought of as the server/process available 99.999 % of time
  - For a large-scale node system, there is a high probability that a node is either down or that there is a network partitioning
- **Partition tolerance**
  - Ensures that write and read operations are redirected to available replicas when segments of the network become disconnected

The Idea of Eventual Consistency

- Eventually Consistency – copies becomes consistent at some later time if there are no more updates to that data item
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
  - For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- BASE (Basically Available, Soft State, Eventual consistency) properties, as opposed to ACID
  - Soft state: copies of a data item may be inconsistent
  - Basically Available – possibilities of faults but not a fault of the whole system
  - Data becomes eventually consistent





State of Practice in Data Storage

Advantage NoSQL

- Application changes dramatically due to four megatrends:
  - The growing number of internet users that applications must support (along with elevated user expectations for how applications should perform).
  - An increase in the volume and variety of data available.
  - A proliferation of machine-generated data from the Internet of Things.
  - A shift to cloud computing, which relies on a distributed three-tier internet architecture.
- As a result, the use of NoSQL technology is increasing.
  - Enterprises need to adapt to new big data management capabilities of modern applications, including:
    - Better application development productivity through a more flexible data model.
    - The ability to scale out dynamically and cost effectively to support more users and big data.
    - Improved performance that satisfies user expectations for highly responsive applications and allows more complex processing of data.
- NoSQL is increasingly considered a viable alternative to relational databases.
  - Considered better particularly for interactive web and mobile applications.

Better Performance, Scalability, & Flexibility

SQL vs NoSQL vs NewSQL

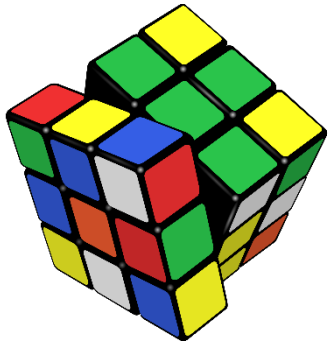
Big Data Engineering For Analytics

NewSQL is a term coined by 451 Group analyst Matt Aslett to describe a new group of databases that share much of the functionality of traditional SQL relational databases, while offering some of the benefits of NoSQL technologies.

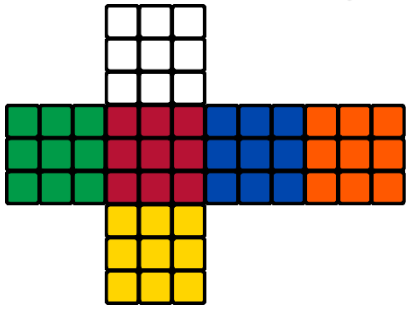
Characteristic	SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
SQL language support	Yes	No*	Yes
Fully ACID Compliant	Yes	No	Yes
Horizontal Scalability (Scale-out vs. Scale-up)	No	Yes	Yes
High Performance	No	Yes	Yes
Schema-on-Write	Yes	No	Yes

Big Data and Cloud

- Big Data Solutions invariably need Cloud based systems (distributed systems)
- This is because:
  - Volume of data is well catered to in the cloud.
  - Large data volume needs greater processing power which is facilitated by cloud.
  - The variety of data may need the use of multiple database products, which is more easily licensed and administered in the Cloud.
  - Cloud system Vendors provide both database and file storage options.
  - Examples:
    - File Storage: Local file system, Network file system, Mounted Volume, Object Storage
    - Database: RDBMS & NoSQL (Document, Key Value, Column Family, Graph
  - Cloud vendors provide a rich set of Analytical and Machine Language tools that is imperative in most Big Data Solutions.



Big Data Engineering For Analytics



Summary

Systematically identity top designers as early as possible. The best are often not the most experienced.

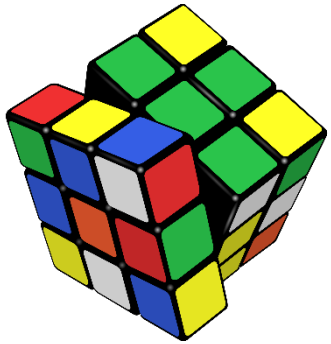
Fred Brooks

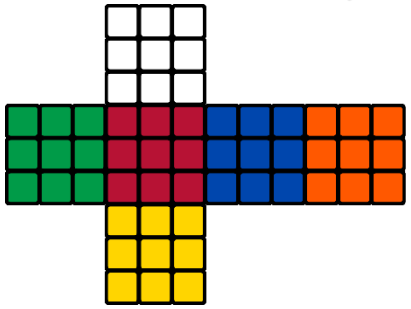
Summary - Characteristics of NoSQL

- Further Classified as
  - **Key/value store** (in-memory, persistent and even ordered) — Redis and BerkeleyDB
  - **Document store** — MongoDB and CouchDB
  - **Column-family-based store** — HBase and Hypertable
  - **Eventually consistent key/value store** — Apache Cassandra and Voldermot
  - **Graph stores** – Neo4j
- Characteristics:
  - Not using the relational model; Schema less Data Stores
  - Running well on clusters; Built for the 21st century web estates
  - Open-source
  - Covers for both structured and unstructured data
  - Results in the rise of NoSQL is Polyglot Persistence

Summary – Key Points.

- Next generation of highly scalable and elastic data stores are the NOSQL databases
  - Designed to scale out horizontally on shared nothing machines
  - Provide Relaxed ACID guarantees (CAP Theorem)
  - Employs a lock-free concurrency control scheme to avoid user shut down,
  - Provides higher performance than available from the traditional systems.
- NoSQL is an abstraction for a class of data stores
  - NoSQL is a concept, a classification, and a new-generation data storage viewpoint.
- Sharding distributes different data across multiple servers, so each server acts as the single source for a subset of data.
- Replication copies data across multiple servers, so each bit of data can be found in multiple places





References

"23 Exabytes of information was recorded and replicated in 2002. We now record and transfer that much information every 7 days."

~ Robert Moore

References

- NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence ,by Martin Fowler; Pramod J. Sadalage, Published by Addison-Wesley Professional, 2012.
- Professional NoSQL, by Shashank Tiwari, Published by Wrox, 2011.
- Getting Started with NoSQL, By Gaurav Vaish, Packt Publishing, 2013.
- Blogs, Articles, White papers and related Internet references.

