Chenega Professional & Technical Services

Headquarters:

609 Independence Parkway
Suite 210
Chesapeake, VA 23320

Phone:

757-549-5700

Web:

www.chenegapts.com

LinkedIn:

www.linkedin.com/company/chenega-professional-services-business-unit/

NoSQL Databases – An Overview

A White Paper by Chenega Professional & Technical Services, LLC

Adam Getz
Solution Architect, Data Analytics
January 2019

What is NoSQL?

NoSQL databases provide an increasingly popular high-performance alternative to traditional relational database management systems (RDBMS) when working with large unstructured, semi-structured data. NoSQL, which stands for "Not Only SQL" (https://en.wikipedia.org/wiki/NoSQL), is unlike traditional RDBMS as it designed for processing large collections of distributed data that don't fit well into strict conventional rows and columns. Known for high-performance, scalability, and flexibility, NoSQL databases have gained substantially in popularity since the advent of Web 2.0.

> The term NoSQL was first used by Carlo Strozzi in 1998 to name his lightweight Strozzi NoSQL open-source relational database that did not expose the standard Structured Query Language (SQL) interface but was still relational.
>
> In early 2009, the term was re-introduced when Johan Oskarsson organized an event to discuss "open source distributed, non-relational databases"
>
> Wikipedia (https://en.wikipedia.org/wiki/NoSQL)

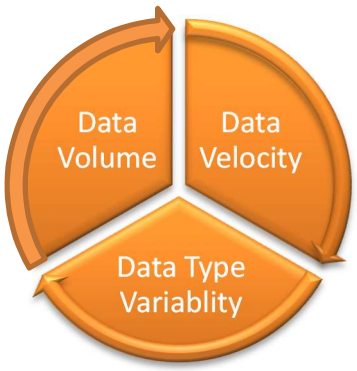
The term **NoSQL** can be applied to some databases that were available before traditional RDBMS, but more often the term refers to databases developed in the mid to late 2000s for the purpose of large-scale database processing within web and mobile based applications. Within these emerging applications, requirements for performance and scalability outweighed the conventional requirement for the rigid data consistency that existing RDBMS provided to transactional applications. Subsequently, NoSQL databases for web applications have tended to focus on very specific characteristics of data management. The ability to process very large volumes of data and quickly distribute that data across computing processors and clusters has been very desirable in large-scale web application design. There has also been a greater need for flexible data schema, or no schema at all, in order to better implement rapid changes to applications.

NoSQL Database Typically Contain the Following Types of Data:

- Semi-structured data (CSV, Word, Excel, PowerPoint, Documents, PDFs, Logs, XML, JSON)
- Unstructured data (Emails, Text, Messages, Blog Entries, Twitter)
- Binary data (Graphics, Images, Audio, Video)

An advantage of **NoSQL** databases over traditional RBMS is that they store and manage data in ways that allow for high operational speed and great flexibility on the part of system developers. In addition, data can be stored in a schema-less or free-form fashion. Any data can be stored in any record. And unlike traditional RDBMS, many NoSQL databases can be scaled horizontally across hundreds or thousands of commodity servers. And NoSQL database typically utilize lower amounts system memory than RDBMS. This allows for NoSQL databases to achieve much higher performance than traditional RDBMS.

Business Rationale for NoSQL



Increases of data volume, data velocity, and data type variability within modern business organizations has created a high demand on conventional relational database management systems (RDBMS) and requires a new paradigm for organizations to remain effective. Organizations have been realizing that they now need to rapidly capture and analyze immensely large amounts of changing data that is being received in many different formats. Data volume and data velocity refer to the ability to process large data sets as they rapidly arrive. Data type variability refers to diversity in data types that don't easily fit into structured database tables. ***Referencing the book, "Making Sense of NoSQL", by Dan McCreary and Ann Kelly.***

- **Data Volume:** Volume refers to the incredible amounts of data generated each second from social media, email, message, text documents, smart phones, sensors, photographs, video, etc. The vast amounts of data have become so large in fact that the data can no longer be stored and analyzed using traditional database technology. Now that data is generated by machines, networks, and human interaction, the amount of data to be analyzed is massive. We now use distributed systems, where parts of the data are stored in different locations and brought together by software. Collecting and analyzing this data is clearly an engineering challenge of immensely vast proportions. More sources of data with a larger size of data combine to increase the amount of data that must be analyzed. This is a major issue for those organizations looking to put that data to use instead of letting it just disappear.
- **Data Velocity:** Velocity refers to the speed at which vast amounts of data are being generated, collected and analyzed. Additional, velocity deals with the pace at which data flows in from sources like business processes, machines, networks and human interaction. And the flow of data is both continuous and massive in amount. Real-time data can help researchers and businesses make valuable & timely decisions that provide both strategic and competitive advantages, as well as a high return on investment (ROI). Not only must the data be rapidly analyzed, but the speed of transmission, and access to the data must also remain instantaneous. In the past, companies analyzed data using long-running batch processes. That paradigm worked well when the incoming data rate was slower than the batch processing rate and when the result was useful despite the delay in analysis execution. With new sources of data such as social, web, and mobile applications, the batch process paradigm has broken down. Now data is now streaming into servers in a real-time, continuous fashion and the result is only useful if data is immediately analyzed with very little delay.

- **Data Type Variability:** Variability refers to the many sources and types of data both structured and unstructured. In the past, data was managed primarily within spreadsheets and relational databases. Now data comes in the form of emails, text, photo, audio, video, web, GPS data, sensor data, relational databases, documents, messages, pdf, flash, etc. Data structures have changed to lose its rigid structure and hundreds of data formats are now being implemented. Organizations no longer have control over the input data format. Structure can no longer be imposed like in the past in order to keep control over the analysis. Organizations that want to capture and report on exception data struggle when attempting to use rigid database schema structures imposed by traditional relational database management systems. More and more, data being created and being analyzed is of the unstructured variety. New and innovative technologies are now allowing both structured and unstructured data to be harvested, stored, and processed simultaneously.

Core Themes of NoSQL Databases

Core Theme	Description
Multiple data formats:	NoSQL databases store and retrieve data from many formats: key-value stores, graph stores, wide column / column family stores, document stores, & search engines.
Free of table joins:	NoSQL databases allow for extraction of data using simple interfaces without the use of joins between tables.
Free of pre-defined schema:	NoSQL databases allow users to place data into a file folder and then query the data without defining a data schema.
Distributed processing:	NoSQL databases can use more than one or multiple computer processors in order to execute.
Horizontal scaling / scaling out:	NoSQL databases have direct increases of system performance with the addition of computer processors.
Design alternatives:	NoSQL databases offer multiple options to a traditional single method of storing, retrieving, and manipulating data.
High performance:	NoSQL database are optimized for specific data models and access patterns that enable higher performance than trying to accomplish similar functionality with relational databases.
Rapid implementations:	NoSQL databases generally provide flexible data schema that enable faster and more iterative development.

Referencing the book, "Making Sense of NoSQL", by Dan McCreary and Ann Kelly

Misconceptions of NoSQL Databases

Misconception	Description
NoSQL is all not about the SQL query language:	NoSQL databases are not applications that utilize a language other than SQL. SQL as well as other query languages are used with NoSQL databases.
NoSQL is not all about open source projects:	Although many NoSQL database are built upon an open source model, commercial products use NoSQL concepts as well as open source initiatives.
NoSQL is not only used in big data projects:	Many NoSQL databases are driven by the inability of an application to efficiently scale when big data is utilized. While data volume and data velocity are important to NoSQL database implementations, NoSQL databases also focus on data type variability and the ability to rapidly implement solutions.
NoSQL is not only used in cloud environments:	Many NoSQL databases do reside in cloud environments to take advantage of the cloud's ability to rapidly scale. But NoSQL databases can run in both the cloud as well as on-premise data centers.
NoSQL is not all about a clever use of memory and SSD:	Many NoSQL databases do focus on the efficient use of computer memory and/or solid-state disks (SSD) to increase performance. While important, NoSQL databases can run on standard commodity hardware.
NoSQL is not just a few products:	More and more NoSQL databases are constantly being developed. And existing NoSQL databases are constantly being enhanced to included additional functionality.
NoSQL databases are not just about solving one problem:	While many NoSQL databases have only been developed using one type of database model, many other NoSQL databases are multi-modal and can solve multiple types of problems.

Referencing the book, "Making Sense of NoSQL", by Dan McCreary and Ann Kelly

Types of NoSQL Databases

According to **Studio 3T** (https://studio3t.com/whats-new/nosql-database-types), NoSQL databases were born out of the rigidity of traditional relational or SQL databases, which use tables, columns, and rows to establish relationships across data. Developers welcomed NoSQL databases because they didn't require an upfront schema design; they were able to go straight to development. And it's this flexibility, this "ad-hoc" approach to organizing data, that has arguably been NoSQL's greatest selling point, which continues to appeal to organizations that need to store, retrieve, and analyze either unstructured or rapidly changing data.

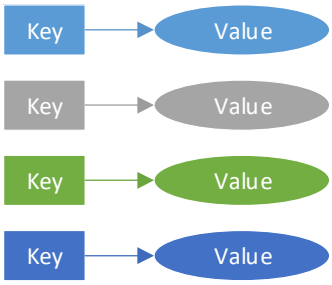
In this paper, we discuss five main types of NoSQL databases, namely key-value store, document store, graph store, wide column / column-family store, and search engine. However, a sixth type of NoSQL database exists that combines aspects of other NoSQL databases. This sixth database type is known as multi-modal as it is a combination of NoSQL types.

Database Type	Description	Typical Uses
Key-Value	A simple data storage system that pairs a unique key with an associated value.	• Dictionaries • Image stores • Document/file stores • Query cache • Lookup tables
Document Store	Data stores that pair each key with a complex data structure known as a document. Documents are typically semi-structured either in XML or JSON formats.	• MS Word documents • MS Excel documents • PDF files • Sales orders • Invoices • Product descriptions • Web pages • Forms
Graph	Data stores that organize data as nodes, which are like records in a relational database, and edges, which represent connections between nodes.	• Social networks • Fraud detection • Pattern matching • Custom relationship management • Law enforcement intelligence

Database Type	Description	Typical Uses
Wide Column / Column Family	Data stores that can hold very large numbers of dynamic columns. But unlike a relational database, the names and format of the columns can vary from row to row in the same table.	• Web crawling • Large sparsely populated tables • Highly-adaptive systems • High-variance systems
Search Engine	Information retrieval systems designed to help find information stored on a computer system.	• Document search • File search • Folder search • Data file search
Multi-Modal	Data stores that contain aspects of multiple types of NoSQL database all within one product.	• Combination of multiple NoSQL aspects

Referencing the book, "Making Sense of NoSQL", by Dan McCreary and Ann Kelly

Key Value NoSQL Database



A **key-value database**, also known as a key-value store, is the most flexible type of NoSQL database. Key-value databases have emerged as an alternative to many of the limitations of traditional relational databases, where data is structured in tables and the schema must be predefined. In a key-value data store, there is no schema and the value of the data can be just about anything. Values are identified and accessed via a key, and values can be numbers, strings, counters, JSON, XML, HTML, binaries, images, videos, and more. It is the most flexible NoSQL model because the application has complete control over what is stored in the value field with no restrictions.

A **key-value database** is a simple database that contains a simple string (the key) that is always unique, and an arbitrary large data field (the value). Key-value stores have no query language, but they do provide a way to add and remove key-value pairs. Additionally, values cannot be queried or searched upon. Only the key can be queried. Within a key-value database, only three functions can be executed (put, get, delete).

- **Put:** Adds a new key-value pair and updates the value if the key is already present.
- **Get:** Returns the value for any given key.
- **Delete:** Removes a key-value pair.

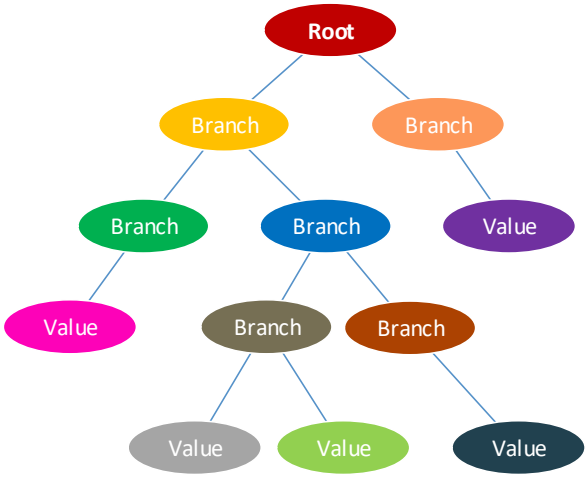
One of the benefits of the **key-value database** is that data of any data type can be stored in the value field including a binary large object (BLOB) value. Additionally, the key-value database is like a dictionary, as a dictionary has a list of words and each word has one or more definitions with various length. And like a dictionary, the key-value database is uniquely indexed by the key field. Thus, rapid retrieval of values can occur regardless of the number of records / items within the database. Key-value databases work in a very different fashion from traditional relational database management systems (RBMS). Traditionally RDBMS define data structures in the database as a series of tables containing fields with well-defined data types. In contrast, key-value databases treat data values as a single opaque collection, which may have different fields for every record. This offers considerable flexibility and more closely follows modern concepts like object-oriented programming. Because optional values are not represented by placeholders or input parameters, key-value databases often use far less memory to store the same database, which can lead to significant performance gains.

Key	Value
user-123	“John Doe”
image-123.jpg	<binary image file>
http://webpage-123.html	<web page html>
file-123.pdf	<pdf document>

Document NoSQL Database

A **document database**, also called a document store or document-oriented database, is used for storing, retrieving, and managing semi-structured data. Unlike traditional relational database management systems (RDBMS), the data model in a document database is not structured in a table format of rows and columns. A document database uses documents as the structure for storage and queries. Subsequently a document database aggregates data from documents and stores the documents a searchable and organized format. The schema of document databases can vary, providing far more flexibility for data modeling than RDBMS. In this case, the term “document” may refer to a MS Word, MS Excel, MS PowerPoint or Adobe PDF document but is commonly a block of extensible markup language (XML) or javascript object notation (JSON) code and values. Instead of columns with names and data types that are used in RDBMS, a document contains a description of the data type and the value for that description. Each document stored within a document database can have the same or different structure.

Documents within **document databases** are identified using a unique key, which contains a simple identifier. The key usually contains either a string, a URI, or a path. And the key can be used to retrieve the document from the database. Typically, the database retains an index on the key to speed up document retrieval. And sometimes, the key is used to create or insert the document into the database.



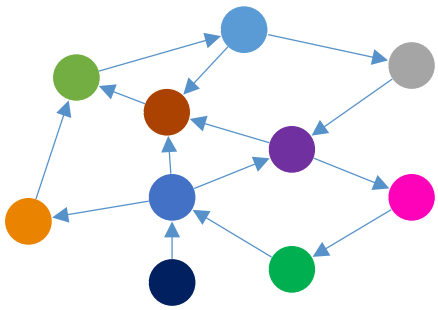
Key	Document
document-1	{ "id": "1", "name": "John Smith", "isactive":"true", "birthdate":"08/30/1984" }
document-2	{ "id": "2", "fullname": "Sara Walker", "isactive":"false", "birthdate":"02/15/1971" }
document-3	{ "id": "3", "fullname": { "firstname":"Max", "lastname":"Johnson", "middleinitial":"B" } "isactive":"true", "birthdate":"04/02/1964" }

Document databases use a tree-like structure that begins with a root node. And beneath the root node, there is a sequence of branches, sub-branches, and values. Subsequently, each branch has a related path expression that shows to navigate from the root of the tree to any given branch, sub-branch, or value. Most document stores group documents together within document collections. And these collections are similar in look and feel to the directory structure in a Windows or UNIX/Linux file system. Document collections can be used to navigate document hierarchies, logically group similar documents, and to store business rules including permissions, indexes, and triggers. Additionally, collections can contain other collections.

A key advantage of a **document database** is that all values within the document are automatically indexed when a new document is insert into the database. That means that every value within the document can be searched upon. This also means that if a user knows any property of the document, all documents with the same property can be easily retrieved. And even if the document structure is complex, a document store search provides an easy way to select either an entire document or a sub-set of a document. Additionally, document database searches can tell the user whether the search item is included within a document as well as the search items exact location utilizing the document path.

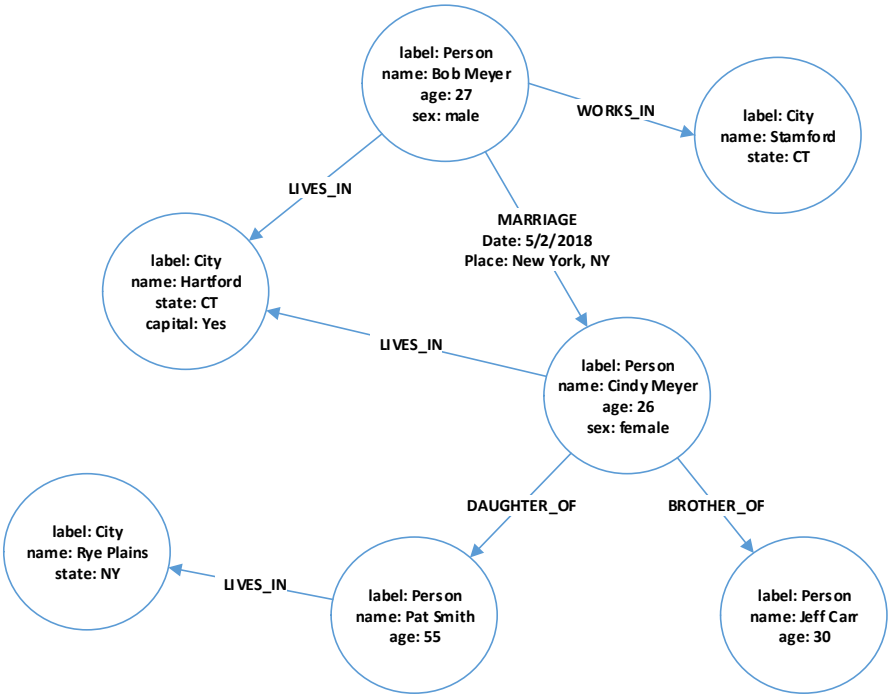
To add additional types of data to a **document database**, there is no need to modify the entire database schema as is with a RDBMS. Data can simply be added by adding objects to the database. Further document databases utilize internal structure within documents in order to extract metadata that the database engine uses for further optimization and query performance. Unlike traditional RDBMS, some document databases prioritize write availability over strict data consistency. This ensures that writes will always be fast even if there is a failure in one portion of the hardware or network.

Graph NoSQL Database



A **graph database** organizes data as nodes, which are like records in a relational database, and relationships, which represent connections between nodes. Because the graph system stores the relationship between nodes, it can support richer representations of data relationships. Relationships are the key concept in graph databases, representing an abstraction that is not directly implemented in RDBMS or other NoSQL databases. Primarily, graph databases are applied in systems that share relationships between values, such as social networks, reservation systems, fraud detection, or customer relationship management systems. And graph databases address significant limitations of existing relational database management systems (RDBMS).

Graph databases, by design, allow simple and fast retrieval of complex hierarchical structures that are difficult to model with RDBMS. They allow for simple queries that display the nearest neighboring nodes. And they allow for complex queries that explore vast networks of connections and quickly find patterns in the connections. Flexible structure enables graph databases to accommodate complex data that doesn't conform to rigid data models required for RDBMS implementations.



Graph databases contain four types of data fields (nodes, relationships, properties, & labels):

- **Nodes:** Objects that represent data entities or instances such as people, businesses, accounts, products or any other item to be tracked. They are roughly the equivalent of the record or row in a relational database, or the document in a document-store database. Each node contains several pieces of information that go together. For example, a single node might include a product name, description, price and product code. Another might have information about a customer, such as name and account number.
- **Relationships:** Objects that describe how the nodes relate to each other. Relationships represent the connections, edges, or lines between nodes to other nodes. A relationship connects two nodes and enables users to find related nodes. A relationship always has a source node and a target node that provides the direction of the arrow. Meaningful patterns can emerge when examining the connections and interconnections of nodes.
- **Properties:** Additional attributes of both nodes and relationships that are represented as additional key-value pairs. Properties store relevant data about the node or relationship with the entity it describes. Examples of priorities for a node with a label of person include name, age, address, & date of birth. Relationships usually have properties including time, distance, cost, rating or weights which are also stored as key-value pairs.
- **Labels:** Named graph construct that is used to group nodes into sets, and all nodes with the same label belongs to the same set. Many database queries can work with these sets instead of the whole graph, making queries easier to write and more efficient to execute. A node may be labeled with any number of labels, including none, making labels an optional addition to the graph.

Each node in the **graph database** model directly and physically contains a list of relationships that represent the connections to other nodes. Unlike traditional RDBMS, graph databases do not utilize foreign keys or join operations. Instead, all relationships are natively stored within vertices.

Graph databases are purpose-built for the analysis of interconnections and relationships of data entities. This design relates well to analysis of data retrieved from social media, web, and mobile applications,. Graph databases are also useful for working with data in business disciplines that involve analyzing complex relationships and dynamic schema, such as supply chain management, customer relationship management, law enforcement intelligence, and fraud detection.

Wide Column / Column Family NoSQL Database

Wide column / column family databases store data in records with an ability to hold very large numbers of dynamic columns. Columns can contain null values and data with different data types. In addition, data is stored in cells grouped in columns of data rather than as rows of data. Columns are logically grouped into column families. Column families can contain a virtually unlimited number of columns that can be created at run-time or while defining the schema. And column families are groups of similar data that is usually accessed together. Additionally, column families can be grouped together as super column families.

	company				
	name	**address**		**website**	
1	DataX	**city**	San Francisco	**protocol**	https
		state	California	**domain**	datax.com
		street num	135	**subdomain**	www
		street	Kearny St		
2	Process-One	**city**	Arlington	**protocol**	https
		state	Virginia	**domain**	process1.com
		street num	3500	**subdomain**	www
		street	Wilson St		

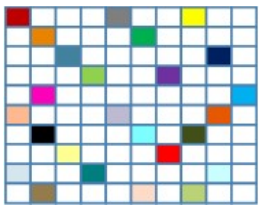
super column family

column

row key

column family

The main advantages of storing data in columns over relational DBMS are fast search/access and data aggregation. Wide column databases store all the cells corresponding to a column as a continuous disk entry, thus making the search/access faster. Wide column databases enable enormous amounts of data to be collected and stored. And wide column databases are purpose-built for **sparse matrix** implementations in which only a small amount of the cells contain values. Its architecture uses persistent, multi-dimensional mapping (row-value, column-value, and timestamp) in a tabular format meant for massive scalability (over and above the petabyte scale).



The basis of the architecture of **wide column / column family databases** is that data is stored in columns instead of rows as in a conventional relational database management system (RDBMS). And the names and format of the columns can vary from row to row in the same table. Subsequently, a wide column database can be interpreted as a two-dimensional key-value. Wide column databases do often support the notion of column families that are stored separately. However, each such column family typically contains multiple columns that are used together, like traditional RDBMS tables. Within a given column family, all data is stored in a

row-by-row fashion, such that the columns for a given row are stored together, rather than each column being stored separately.

Since **wide column / column family databases** do not utilize table joins that are common in traditional RDMS, they tend to scale and perform well even with massive amounts of included data. And databases with billions of rows and hundreds or thousands of columns are common. For example, a geographic information systems (GIS) like Google Earth may a row ID for every longitude position on the planet and a column for every latitude position. Thus, if one database contains data on every square mile on Earth, there could be thousand of rows and thousands of columns in the database. And most of the columns in the database will have no value, meaning that the database is both large and sparsely populated.

Search Engine NoSQL Database

Search engine databases deal with data that does not necessarily conform to the rigid structural requirements of relation database management systems (RDBMS) as data for search may be text-based, semi-structured, or unstructured. Search engine databases are made to help users quickly find information they need in a high-quality and cost-effective manner. They are optimized for key word queries and typically offer specialized methods such as full-text search, complex search expressions, and ranking of search results.

Search engine databases contain two main components. First content is added to the search engine database index. Then when a user executes a query, relevant results are rapidly returned utilizing the search engine database index. Fast search responses are possible because instead of searching the text directly, queries perform searches against an index. This is the equivalent of retrieving pages in a book related to a keyword by searching the index at the back of a book, as opposed to searching each of the words in each page of the book. This type of index is known as an inverted index, because it converts a page-centric data structure to a keyword-centric data structure.

Search engine databases commonly support the following types of search functionality.

- **Full-text search:** Compares every word of the search request against every word within a file. Examines all the words in every stored file that contains natural language text such as English, French, or Spanish. And is appropriate when data to be discovered is mostly free-form text like that of a news article, academic paper, essay, or book.
- **Semi-structured search:** Searches of data that have both the rigid structure of an RDBMS and full-text sentences like those in a MS Word or PDF document as they can be converted to either XML or JSON format. Semi-structured data is a form of data that has a self-describing structure and contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data.
- **Geographic search:** Associates locations to web resources in order to answer location-based queries. Search results will not only be related to the topic of a query, but they will also be related to a physical location associated with the query. Thus, physical locations will be retrieved are in proximity of the search topic.
- **Network search:** Offers a relationship-oriented approach to search that lets users explore the connections in data within stored documents. This can include linkages between people, places, preferences, & products and is useful in discovering relevance of relationships. The search engine processes natural language queries to return information from across network graphs
- **Navigational search:** Augments other search capabilities with a guided-navigation system allowing users to narrow down search results by applying multiple filters based on classification of items. Navigational search uses a hierarchy structure or taxonomy of categories to enable users to browse information by choosing from a pre-determined set of categories. This allows a user to type in a simple query, then refine their search options by either navigating or drilling down into a category.
- **Vector search:** Ranks document results based upon how close they are to search keywords utilizing multi-dimensional vector distance models. Vector search is a way to conduct “fuzzy search”, i.e. a way to find documents that are close to a keyword. They help find inexact matches to documents that are “in-the-neighborhood” of search keywords.

Example NoSQL Databases

Database Type	Examples
Key-Value	• Redis • Amazon DynamoDB • Memchache • MS Azure Cosmos DB • HazelCast • Oracle Berkeley DB
Document Store	• MongoDB • Amazon DynamoDB • Couchbase • CouchDB • MarkLogic • Firebase Realtime Database • OrientDB
Graph	• Neo4j • MS Azure Cosmos DB • DataStax Enterprise • Amazon Neptune
Wide Column / Column Family	• Cassandra • Hbase • MS Azure Cosmos DB • DataStax Enterprise • Accumulo • AllegroGraph • Google Cloud BigTable
Search Engine	• ElasticSearch • Splunk • Apache Solr • MarkLogic • Sphinx • MS Azure Search • Google Search Appliance • Amazon CloudSearch
Multi-Modal	• Amazon DynamoDB • MS Azure Cosmos DB • MarkLogic • DataStax Enterprise • OrientDB

About Chenega Professional & Technical Services

Chenega Professional & Technical Services, LLC (CPTS) is a leading information technology services firm providing consulting and advisory services to clients within the federal government. CPTS specializes in building and implementing solutions that provide both rapid and long-term value in the areas of:

- Data Analytics
- Data Warehousing
- Business Intelligence

Our consultants are both experienced and knowledgeable in the business and technology aspects of IT systems implementations. Plus, we have high degree of subject matter expertise in the areas that are critical to many federal agencies. CPTS is the right organization to deploy the enabling technologies that help federal agencies make more informed decisions through smart use of their data.

CPTS is also a wholly-owned subsidiary of the Chenega Corporation. CPTS utilizes shared services provided by the corporation, which affords us the stability and support of a large business while maintaining the flexibility and rates of a small business. Chenega Corporation has the dual mission of succeeding in business to assist its shareholder, descendants and family members in their journey to economic and social self-determination and self-sufficiency, and to create and support comprehensive cultural, societal and community activities.

For more information visit us at either www.chenegapts.com or www.linkedin.com/company/chenega-professional-services-business-unit.