Questions on Module 3

Q1. List and explain core business drivers behind the NoSql movement.

NoSQL is a type of database management system that focuses on offering great scalability, performance, availability, and agility. It allows for the storage and retrieval of huge amounts of unstructured data in a distributed environment on virtual servers.

Relational databases are not designed to scale and change easily to cope up with the needs of the modern industry. Also, they do not take advantage of the cheap storage and processing power available today by using commodity hardware.

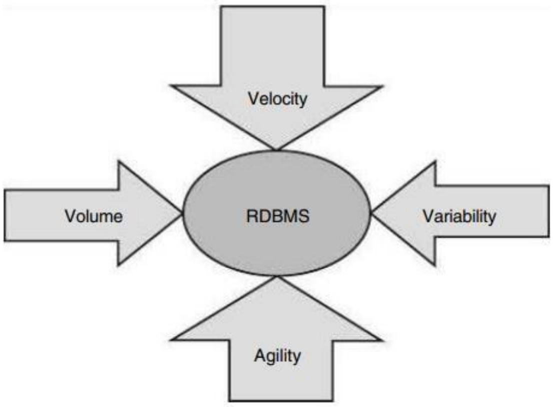
NoSQL database is referred as Not only SQL. Most NoSQL systems are entirely non-relational. They do not have fixed schemas or JOIN operations. They use objects, key-value pairs, or tuples.

Some features of NoSql are

- ✓ NoSQL is non-relational database, and it is schema-free.
- ✓ NoSQL is free of JOINs.
- ✓ NoSQL uses distributed architecture and works on multiple processors to give high performance.
- ✓ NoSQL databases are horizontally scalable.
- ✓ Many open-source NoSQL databases are available.
- ✓ Data file can be easily replicated. 9. NoSQL uses simple API.
- ✓ NoSQL can manage huge amount of data.
- ✓ NoSQL can be implemented on commodity hardware.

NoSQL may not provide atomicity, consistency, isolation, durability (ACID) properties but guarantees eventual consistency, **basically available, soft state** (**BASE**), by having a distributed and fault-tolerant architecture.

Organisations need highly reliable, scalable and available data storage across a configurable set of systems that act as storage nodes. The needs of organizations are changing rapidly. Many organizations operating with single CPU and Relational database management systems (RDBMS) were not able to cope up with the speed in which information needs to be extracted.



NoSql Business Drivers

Businesses have to capture and analyze a large amount of variable data, and take immediate actions/decisions based on the findings after analysis.

Business drivers velocity, volume, variability and agility necessitates the emergence of NoSQL solutions.

All of these drivers apply pressure to single CPU relational model and eventually make the system less stable.

Volume: The key factor pushing organizations to look at alternatives to their current RDBMSs is a need to query big data using clusters of commodity processors. Earlier performance concerns were resolved by purchasing faster processors. The need to scale out (also known as horizontal scaling), rather than scale up (faster processors), moved organizations from serial to parallel processing where data problems are split into separate paths and sent to separate processors to divide and conquer the work.

Velocity: One of the problem with traditional RDBMS systems wrt to Big data is the inability of a single processor system to rapidly read and write data. Many single-processor RDBMSs are unable to keep up with the demands of real-time inserts and online queries to the database made by public-facing websites. When single-processor RDBMSs are used as a back end to a web store front, the random bursts in web traffic slow down response for everyone, and tuning these systems can be costly when both high read and write throughput is desired.

Variability: Companies that want to capture and report on exception data struggle when attempting to use rigid database schema structures imposed by RDBMSs. For example, if a business unit wants to capture a few custom fields for a particular customer, all customer rows within the database need to store this information even though it doesn't apply. Adding new columns to an RDBMS requires the system be shut down and ALTER TABLE commands to be run. When a database is large, this process can impact system availability, costing time and money.

Agility: Building applications with relational databases is complex due to the need to manage data input and output, especially with nested and repeated data structures. An object-relational mapping (ORM) layer is required to generate the necessary SQL statements (INSERT, UPDATE, DELETE, SELECT) for moving data between the application and the database. This process is difficult and often slows down development and testing, even for experienced developers. These challenges can drive developers to consider NoSQL databases, which natively handle nested and hierarchical data structures, simplifying data management and reducing the need for complex ORM layers.

This is how velocity, volume, variability, and agility drives traditional RDBMS systems to use of NoSQL Databases.

Q 2. Describe the CAP theorem of NoSQL database

A relational database product can cater to a more predictable, structured data. NoSQL is required because today's industry needs a very agile system that can process unstructured and unpredictable data dynamically. NoSQL is known for its high performance with high availability, rich query language, and easy scalability which fits the need. NoSQL may not provide atomicity, consistency, isolation, durability (ACID) properties but guarantees eventual consistency.

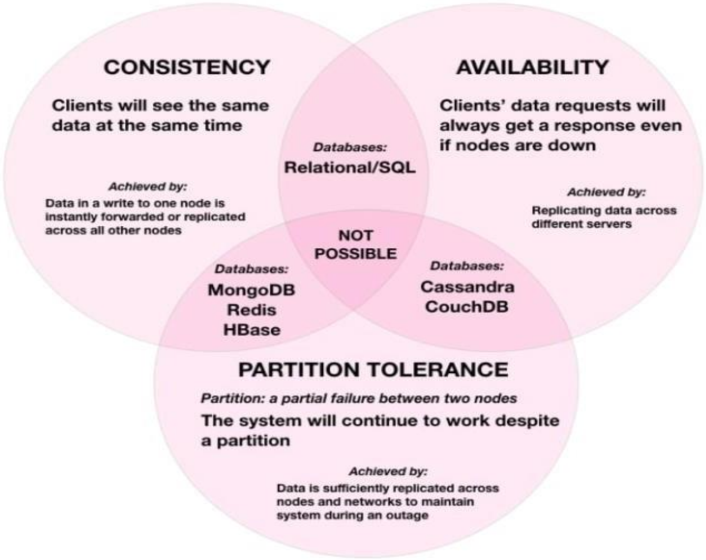
Consistency, Availability , Partition tolerance (CAP) theorem, also called as Brewer's theorem, states that it is not possible for a distributed system to provide all three of the following guarantees simultaneously:

Consistency guarantees all storage and their replicated nodes have the same data at the same time.

Availability means every request is guaranteed to receive a success or failure response. However, it does not guarantee that a read request returns the most recent write. The more number of users a system can cater to better is the availability

Partition tolerance guarantees that the system continues to operate in spite of arbitrary partitioning due to network failures. Partition Tolerance is a guarantee that the system continues to operate despite arbitrary message loss or failure of part of the system. In other words, even if there is a network outage in the data center and some of the computers are unreachable, still the system continues to perform.

CAP theorem or Eric Brewers theorem states that any distributed database can only achieve at most two out of three guarantees for a database: Consistency, Availability, and Partition Tolerance.


CONSISTENCY
Clients will see the same data at the same time
Achieved by:
Data in a write to one node is instantly forwarded or replicated across all other nodes
AVAILABILITY
Clients' data requests will always get a response even if nodes are down
Achieved by:
Replicating data across different servers
Databases:
Relational/SQL
NOT POSSIBLE
Databases:
MongoDB
Redis
HBase
Databases:
Cassandra
CouchDB
PARTITION TOLERANCE
Partition: a partial failure between two nodes
The system will continue to work despite a partition
Achieved by:
Data is sufficiently replicated across nodes and networks to maintain system during an outage

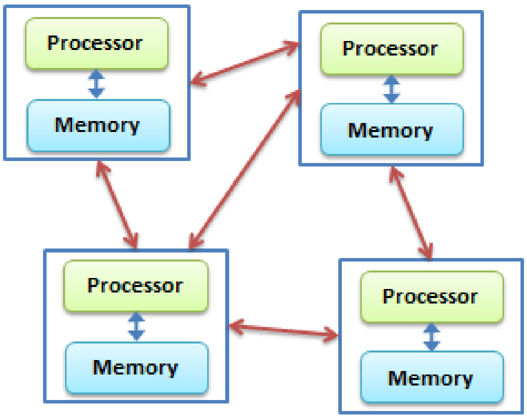
Q3. Name three ways in which resources can be shared between computer systems. Name the architecture used in Big Data Solution.

For massive data processing Parallel and Distributed Systems can be used.

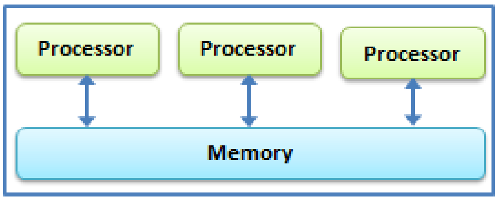
Parallel database system is a tightly coupled system. The processors co-operate for query processing. The processors have access to common memory.

Distributed Systems are known to be loosely coupled and are composed of individual machines. Each machines can run their individual application and serve their own respective user. The data could be distributed across several machines, this makes it necessary to access number of machines to answer a user query.

Distributed Computing

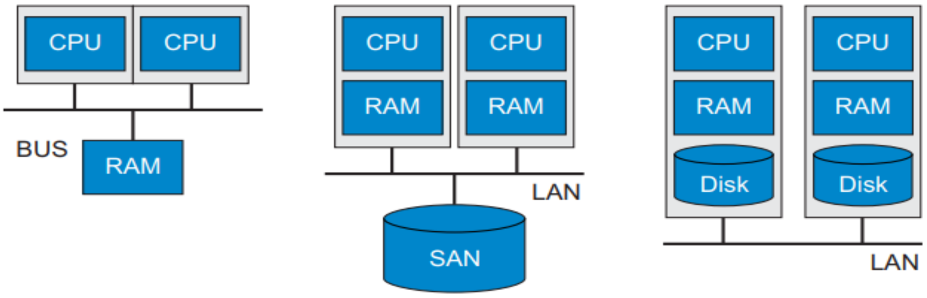


Parallel Computing



Most common types of architecture for resource sharing are

1. Shared Memory (SM)
2. Shared Disk (SD)
3. Shared Nothing (SN)



Shared RAM

Shared disk

Shared-nothing

In **Shared Memory Architecture**, a common central memory is shared by multiple processors.

In **Shared Disk Architecture**, multiple processors share a common collection of disks while having their own private memory.

In **Shared Nothing Architecture**, neither memory nor disk is shared among multiple processors.

Shared Nothing Architecture provides the benefits of isolating fault. A fault in single node is contained and confined to that node exclusively.

Shared Nothing Architecture also makes it possible to scale the architecture thereby allowing to add more systems to the architecture.

Shared Nothing Architecture is best suited for Big Data Solutions.

Q4 List Various NoSQL architectural patterns

A data architecture pattern is a consistent way of representing data in a regular structure that will be stored in memory (HDD or SDD).

The architectural pattern followed by RDBMS is Row Store - where row is a unit of transaction – rows are added, deleted , retrieved as a unit of transaction.

Some systems use columns as an atomic unit of storage. These systems are called column stores.

Column-store systems are used when using aggregates (counts, sums, and so on) and reporting speed is a priority over performance.

There are four types NoSql databases. Each of them has their own attributes and limitations. One single solution cannot be considered better among others as they are meant to solve specific problems.

Data Model	**Performance**	**Scalability**	**Flexibility**	**Functionality**
Key Value	High	High	High	Variable
Column Oriented	High	High	Moderate	Minimal
Document	High	Variable (High)	High	Variable (Low)
Graph	Variable	Variable	High	Graph Theory

Q5 What is Key-Value store? What are the benefits of using Key-Value Store?

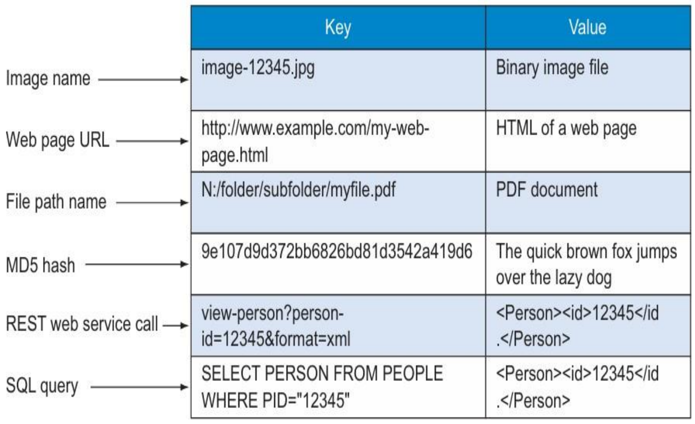
A key-value store is a simple database that when presented with a simple string (the key) returns an arbitrary large BLOB of data (ie the value).

Key-value stores have no query language but they provide a way to add and remove key-value pairs into/from a database.

- ✓ A key-value store is like a dictionary.
- ✓ A dictionary has a list of words and each word has one or more definitions.
- ✓ The dictionary is a simple key-value store where word entries represent keys and definitions represent values.
- ✓ Dictionary entries are sorted alphabetically by word, retrieval is quick; it's not necessary to scan the entire dictionary to find what you're looking for.
- ✓ Like the dictionary, a key-value store is also indexed by the key here the key points directly to the value, resulting in rapid retrieval, regardless of the number of items in your store.
- ✓ One of the benefits of not specifying a data type for the value of a key-value store is that you can store any data type that you want in the value.

- ✓ The system will store the information as a BLOB and return the same BLOB when a GET (retrieval) request is made.
- ✓ It's up to the application to determine what type of data is being used, such as a string, XML file, or binary image.

Key-value stores



	Key	Value
Image name	image-12345.jpg	Binary image file
Web page URL	http://www.example.com/my-web-page.html	HTML of a web page
File path name	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call	view-person?person-id=12345&format=xml	<Person><id>12345</id.</Person>
SQL query	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id.</Person>

The keys can be:

- ✓ Logical path names to images or files
- ✓ Artificially generated strings created from a hash of the value
- ✓ REST web service calls
- ✓ SQL queries
- ✓

Values like keys, are also flexible can be any BLOB object

- ✓ Images
- ✓ Web Pages
- ✓ Documents
- ✓ Videos.

The best way to think about using a key-value store is to visualize a single table with two columns. The first column is called the key and the second column is called the value. There are three operations performed on a key-value store: **put**, **get**, and **delete**. These three operations form the basis of how programmers interface with the key-value store. We call this set of programmer interfaces the **application program interface** or **API**.

A key-value store has two rules:

1. **Distinct keys**—You can never have two rows with the same key-value. This means that all the keys in any given key-value store are unique.
2. **No queries on values**—You can't perform queries on the values of the table.

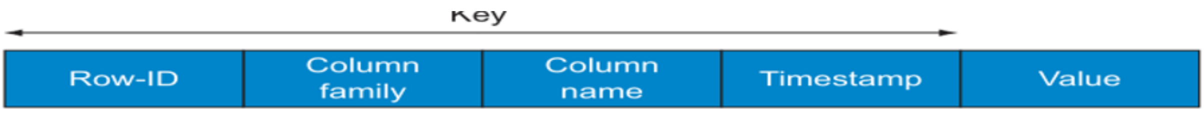
Example NoSql Databases which use Key Value Architecture – Amazon DynamoDB, Riak

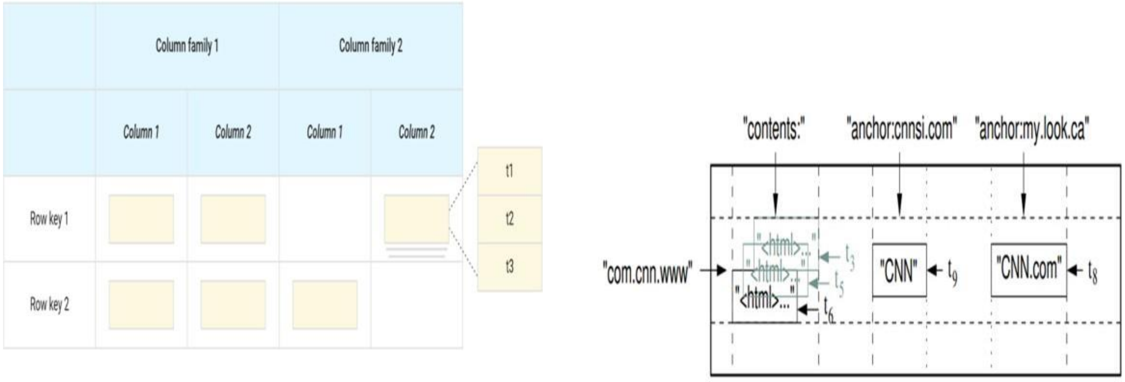
Q6 Explain in detail Column Family Store/ Write a note on Column Family store.

Column family stores use row and column identifiers as general purposes keys for data lookup. They're also referred to as data stores.

A column store database stores all information within a column of a table at the same location on disk in the same way a row-store keeps row data together. Column stores are used in many OLAP systems because their strength is rapid column aggregate calculation.

The key structure in column family store is similar to a spreadsheet but has two additional attributes. In addition to the column name, a column family is used to group similar column names together. The addition of timestamp in the key also allows each cell in the table to store multiple versions of values over time.





Column Family Store

Advantages of column family store

- ✓ **Higher Scalability**
- ✓ **Higher Availability**
- ✓ **Easy to add new data**

Examples : Google Big Table, Cassandra

Q7 What is graph store? Give an example where graph store can be used effectively to solve a particular business problem.

Graph stores are important in applications that need to analyze relationships between objects. A graph store is a system that contains a sequence of nodes and relationships that, when combined, create a graph.

A graph store has three data fields: **nodes**, **relationships**, and **properties.**

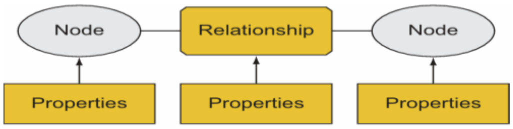


Figure 4.10 A graph store consists of many node-relationship-node structures. Properties are used to describe both the nodes and relationships.

Nodes are usually representations of a distinct real-world objects.

A Node can represent virtually any entity that has a relation with another entity.

Nodes can be people, cars, organizations, telephone numbers, or even more abstract things like ideas.

A Node representing an entity is marked with a unique identifier analogous to a primary key in a relational databases.

Edges also known as a **relationships** or **arcs** represent connections between these objects or show how one object influence on others.

There are two types of edges: directed and undirected. Directed edges have, which should not be surprising, a direction.

This is used to indicate how the relationship, as modeled by the edge, should be interpreted.

For example, in a family relations graph the *parent* relation is one way and should have a direction while *sibling* may stay undirected because is true for both directions.

They are typically represented as arcs (lines) or arrows between circles in diagrams.

Properties describes features (additional information) related to nodes or edge.

A very commonly used property is called the weight of an edge. Weights represent some value about the relationship; the strength of the influence of one object on the other. In general, weights can represent cost, distance, or any other measure of the relationship between objects represented by vertices.

Give an example where graph store can be used effectively to solve a particular business problem

Graph Stores are ideal when there are many items that are related to each other in complex ways and these relationships have properties.

Graph stores allow you to do simple queries that show you the nearest neighboring nodes as well as queries that look deep into networks and quickly find patterns.

For example, if you use a relational database to store your list of friends, you can produce a list of your friends sorted by their last name. But if you use a graph store, you can not only get a list of your friends by their last name, you can also get a list of which friends are most likely to join you on a road trip.

Graph queries are similar to traversing nodes in a graph. You can query to ask things like these:

- ✓ *What's the shortest path between two nodes in a graph?*
- ✓ *What nodes have neighboring nodes that have specific properties?*
- ✓ *Given any two nodes in a graph, how similar are their neighboring nodes?*
- ✓ *What's the average connectedness of various points on a graph with each other?*