# Unit - 1

Overview and History of NoSQL Databases - Definition of the Four Types of NoSQL Database - The Value of Relational Databases: Getting at Persistent Data, Concurrency, Integration - Impedance Mismatch - Application and Integration Databases - Attack of the Clusters - The Emergence of NoSQL - Key Points.

## ❖ Introduction to databases

**What is Data?** : In simple words, data can be facts related to any object in consideration. For example, your name, age, height, weight, etc. are some data related to you. A picture, image, file, pdf, etc. can also be considered data.

**What is Database? :** A database is a systematic collection of data. They support electronic storage and manipulation of data. Databases make data management easy. Let us discuss a database example: An online telephone directory uses a database to store data of people, phone numbers, and other contact details. Your electricity service provider uses a database to manage billing, client-related issues, handle fault data, etc. Let us also consider Facebook. It needs to store, manipulate, and present data related to members, their friends, member activities, messages, advertisements, and a lot more. We can provide a countless number of examples for the usage of databases.

### Types of Databases

**Distributed databases:** A distributed database is a type of database that has contributions from the common database and information captured by local computers. In this type of database system, the data is not in one place and is distributed at various organizations.

**Relational databases:**This type of database defines database relationships in the form of tables. It is also called Relational DBMS, which is the most popular DBMS type in the market. Database example of the RDBMS system include MySQL, Oracle, and Microsoft SQL Server database.

**Object-oriented databases:** This type of computers database supports the storage of all data types. The data is stored in the form of objects. The objects to be held in the database have attributes and methods that define what to do with the data. PostgreSQL is an example of an object-oriented relational DBMS.

**Centralized database:** It is a centralized location, and users from different backgrounds can access this data. This type of computers databases store application procedures that help users access the data even from a remote location.

**Open-source databases:** This kind of database stored information related to operations. It is mainly used in the field of marketing, employee relations, customer service, of databases.

**Cloud databases:** A cloud database is a database which is optimized or built for such a virtualized environment. There are so many advantages of a cloud database, some of which can pay for storage capacity and bandwidth. It also offers scalability on-demand, along with high availability.

**Data warehouses:** Data Warehouse is to facilitate a single version of truth for a company for decision making and forecasting. A Data warehouse is an information system that contains historical and commutative data from single or multiple sources. Data Warehouse concept simplifies the reporting and analysis process of the organization.

**NoSQL databases:** NoSQL database is used for large sets of distributed data. There are a few big data performance problems that are effectively handled by relational databases. This type of computers database is very efficient in analyzing large-size unstructured data.

**Graph databases:** A graph-oriented database uses graph theory to store, map, and query relationships. These kinds of computers databases are mostly used for analyzing interconnections. For example, an organization can use a graph database to mine data about customers from social media.

**OLTP databases**: OLTP another database type which able to perform fast query processing and maintaining data integrity in multi-access environments.

**Personal database:** A personal database is used to store data stored on personal computers that are smaller and easily manageable. The data is mostly used by the same department of the company and is accessed by a small group of people.

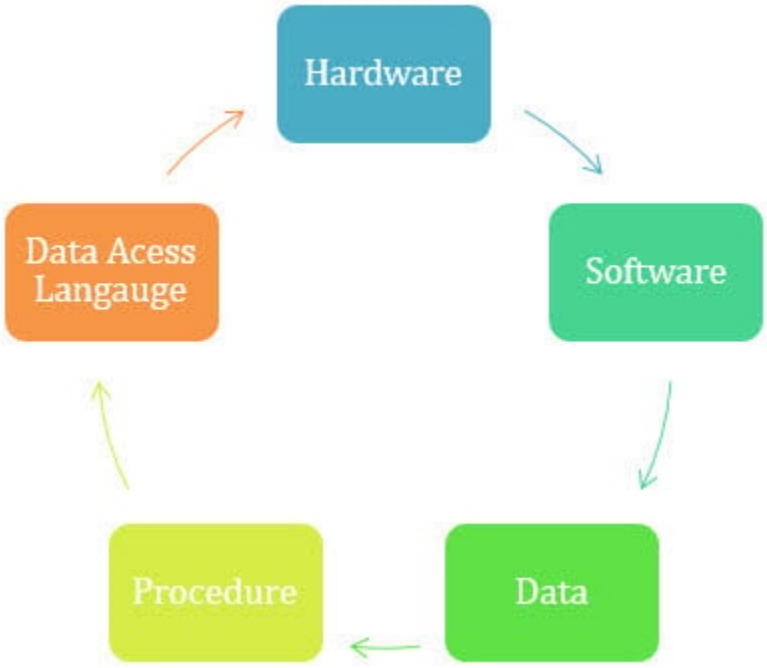
**Multimodal database:** The multimodal database is a type of data processing platform that supports multiple data models that define how the certain knowledge and information in a database should be organized and arranged.

**Document/JSON database:** In a document-oriented database, the data is kept in document collections, usually using the XML, JSON, BSON formats. One record can store as much data as you want, in any data type (or types) you prefer.

**Hierarchical:** This type of DBMS employs the "parent-child" relationship of storing data. Its structure is like a tree with nodes representing records and branches representing fields. The windows registry used in Windows XP is a hierarchical database example.

**Network DBMS:** This type of DBMS supports many-to-many relations. It usually results in complex database structures. RDM Server is an example of database management system that implements the network model.

**Database Components**




**Database Components**
There are five main components of a database:

**Hardware:** The hardware consists of physical, electronic devices like computers, I/O devices, storage devices, etc. This offers the interface between computers and real-world systems.

**Software:** This is a set of programs used to manage and control the overall database. This includes the database software itself, the Operating System, the network software used to share the data among users, and the application programs for accessing data in the database.

**Data:** Data is a raw and unorganized fact that is required to be processed to make it meaningful. Data can be simple at the same time unorganized unless it is organized. Generally, data comprises facts, observations, perceptions, numbers, characters, symbols, images, etc.

**Procedure:** Procedure are a set of instructions and rules that help you to use the DBMS. It is designing and running the database using documented methods, which allows you to guide the users who operate and manage it.

**Database Access Language:** Database Access language is used to access the data to and from the database, enter new data, update already existing data, or retrieve required data from DBMS. The user writes some specific commands in a database access language and submits these to the database.

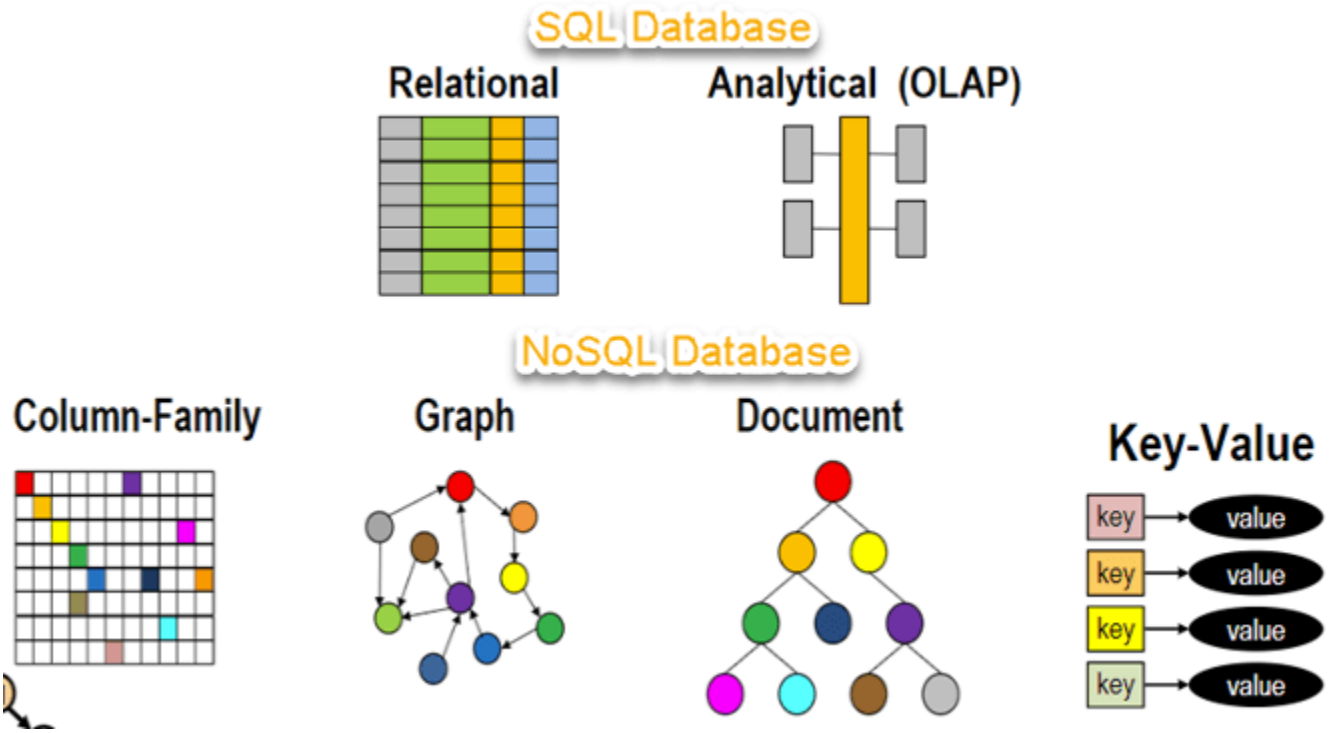
**What is a Database Management System (DBMS)?**

Database Management System (DBMS) is a collection of programs that enable its users to access databases, manipulate data, report, and represent data. It also helps to control access to the database. Database Management Systems are not a new concept and, as such, had been first implemented in the 1960s. Charles Bachman's Integrated Data Store (IDS) is said to be the first DBMS in history. With time database, technologies evolved a lot, while usage and expected functionalities of databases increased immensely.

## ❖ Overview of NoSQL

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

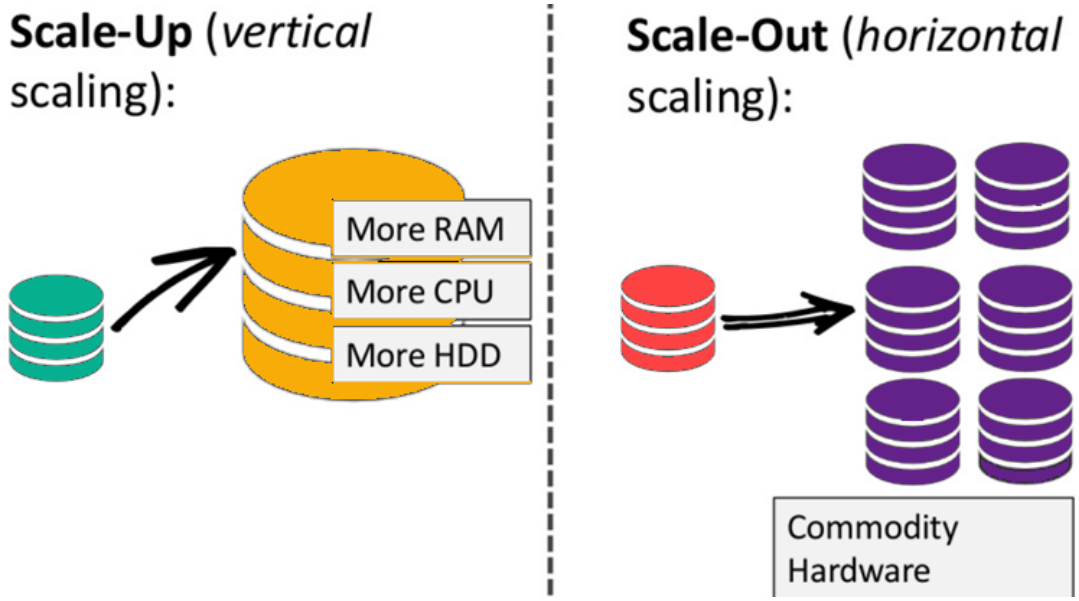
NoSQL database stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.
Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data. Let's understand about NoSQL with a diagram in this NoSQL database tutorial:




Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.
To resolve this problem, we could “scale up” our systems by upgrading our existing hardware. This process is expensive.

> The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as “scaling out.”




NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

Brief History of NoSQL Databases

- 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
- 2000- Graph database Neo4j is launched
- 2004- Google BigTable is launched
- 2005- CouchDB is launched
- 2007- The research paper on Amazon Dynamo is released
- 2008- Facebooks open sources the Cassandra project
- 2009- The term NoSQL was reintroduced

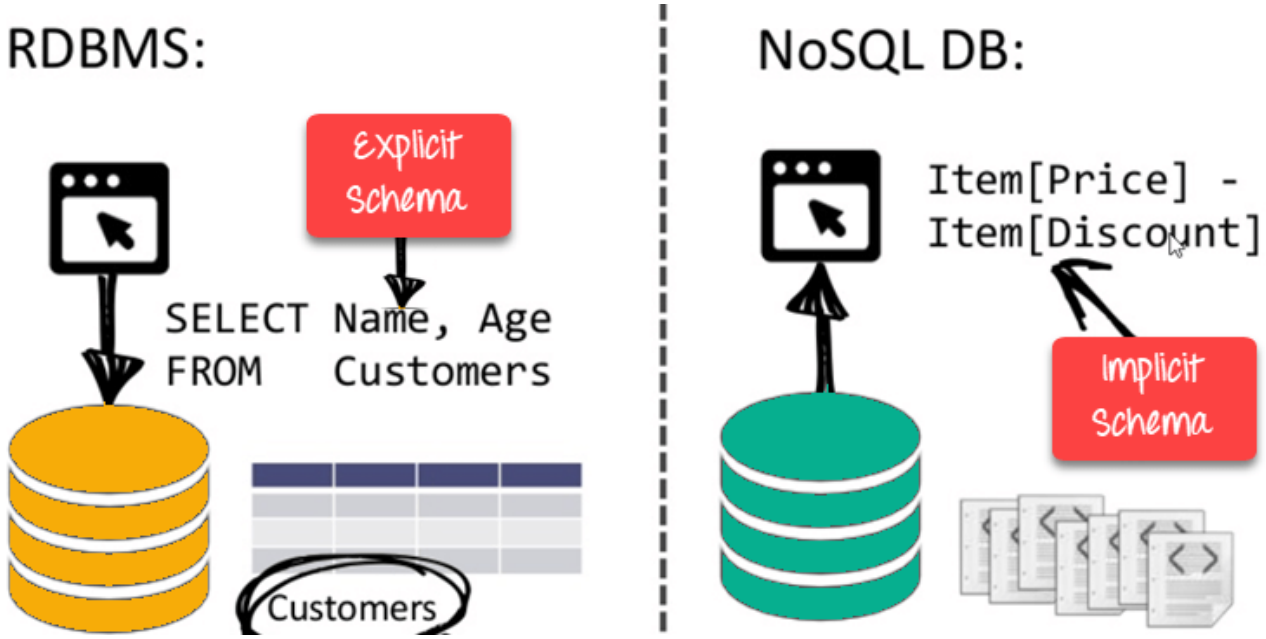
Features of NoSQL

Non-relational

- NoSQL databases never follow the relational model
- Never provide tables with flat fixed-column records
- Work with self-contained aggregates or BLOBs
- Doesn’t require object-relational mapping and data normalization
- No complex features like query languages, query planners,referential integrity joins, ACID

Schema-free

- NoSQL databases are either schema-free or have relaxed schemas
- Do not require any sort of definition of the schema of the data
- Offers heterogeneous structures of data in the same domain



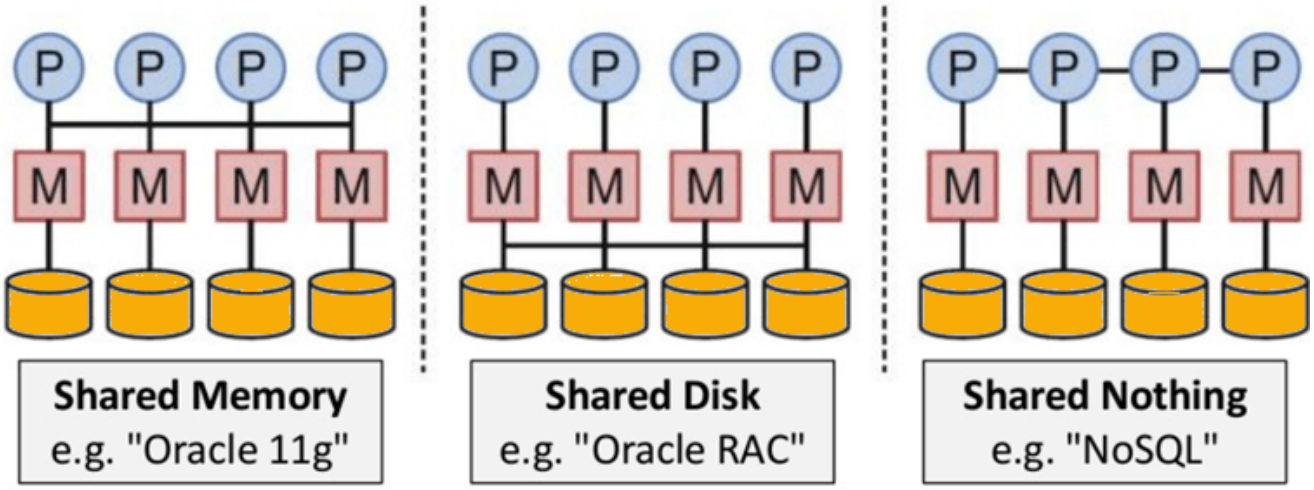

NoSQL is Schema-Free

Simple API

- Offers easy to use interfaces for storage and querying data provided
- APIs allow low-level data manipulation & selection methods
- Text-based protocols mostly used with HTTP REST with JSON
- Mostly used no standard based NoSQL query language
- Web-enabled databases running as internet-facing services

Distributed

- Multiple NoSQL databases can be executed in a distributed fashion
- Offers auto-scaling and fail-over capabilities
- Often ACID concept can be sacrificed for scalability and throughput
- Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
- Only providing eventual consistency
- Shared Nothing Architecture. This enables less coordination and higher distribution.

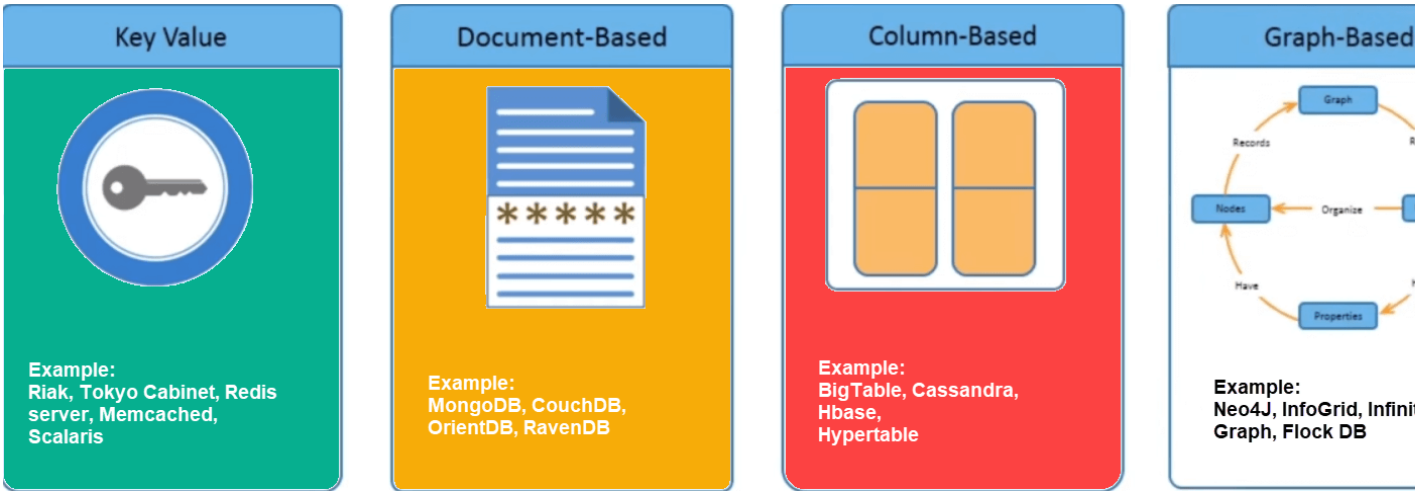



NoSQL is Shared Nothing.

**Types of NoSQL Databases**

NoSQL Databases are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.
Types of NoSQL Databases:

- Key-value Pair Based
- Column-oriented Graph
- Graphs based
- Document-oriented




**Key Value Pair Based**

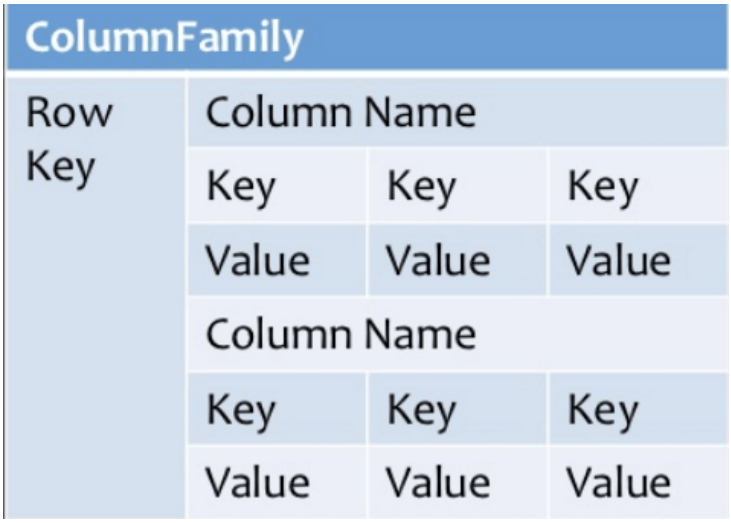
Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.
Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.
For example, a key-value pair may contain a key like “Website” associated with a value like “Guru99”.

| Key | Value |
|---|---|
| Name | Joe Bloggs |
| Age | 42 |
| Occupation | Stunt Double |
| Height | 175cm |
| Weight | 77kg |

It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.
Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon’s Dynamo paper.

**Column-based**

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.




Column based NoSQL database

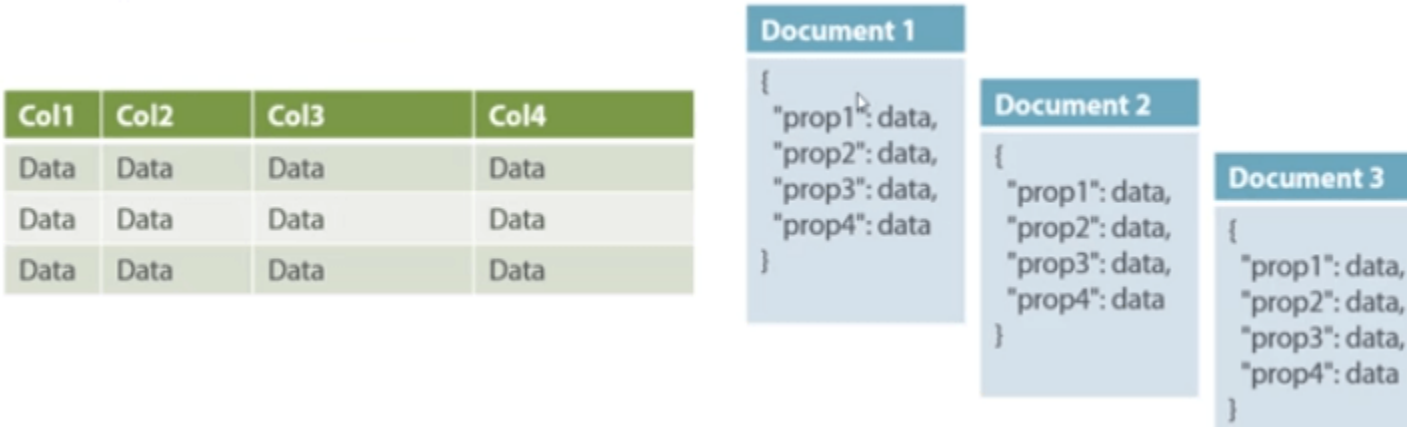
They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.

**Document-Oriented:**

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.




Relational Vs. Document

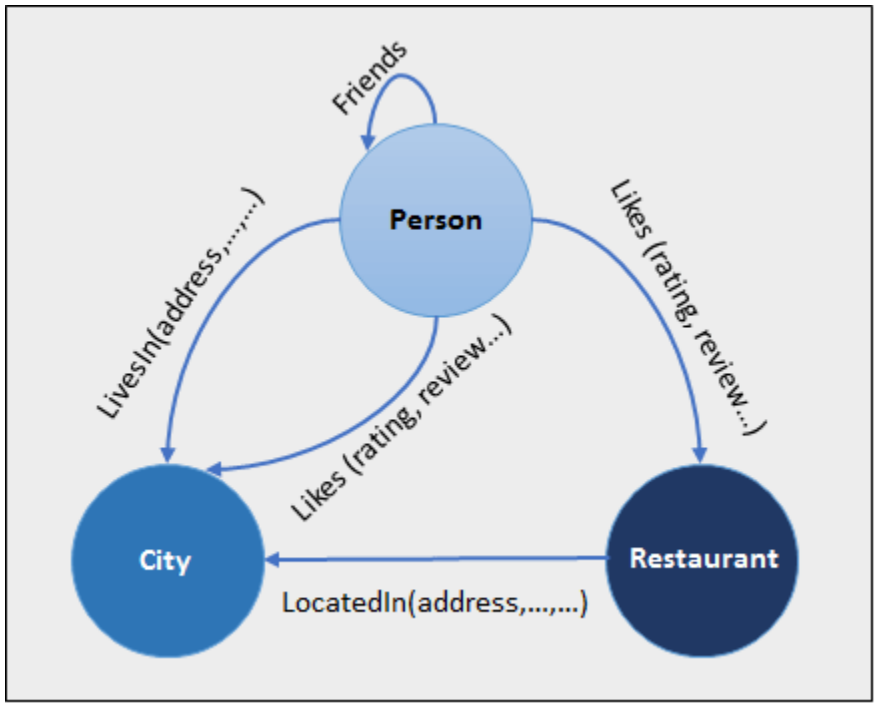
In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

**Graph-Based**

A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.




Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.
Graph base database mostly used for social networks, logistics, spatial data.
Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

**Query Mechanism tools for NoSQL**

The most common data retrieval mechanism is the REST-based retrieval of a value based on its key/ID with GET resource
Document store Database offers more difficult queries as they understand the value in a key-value pair. For example, CouchDB allows defining views with MapReduce

**What is the CAP Theorem?**

CAP theorem is also called brewer's theorem. It states that is impossible for a distributed data store to offer more than two out of three guarantees

1. Consistency
2. Availability
3. Partition Tolerance

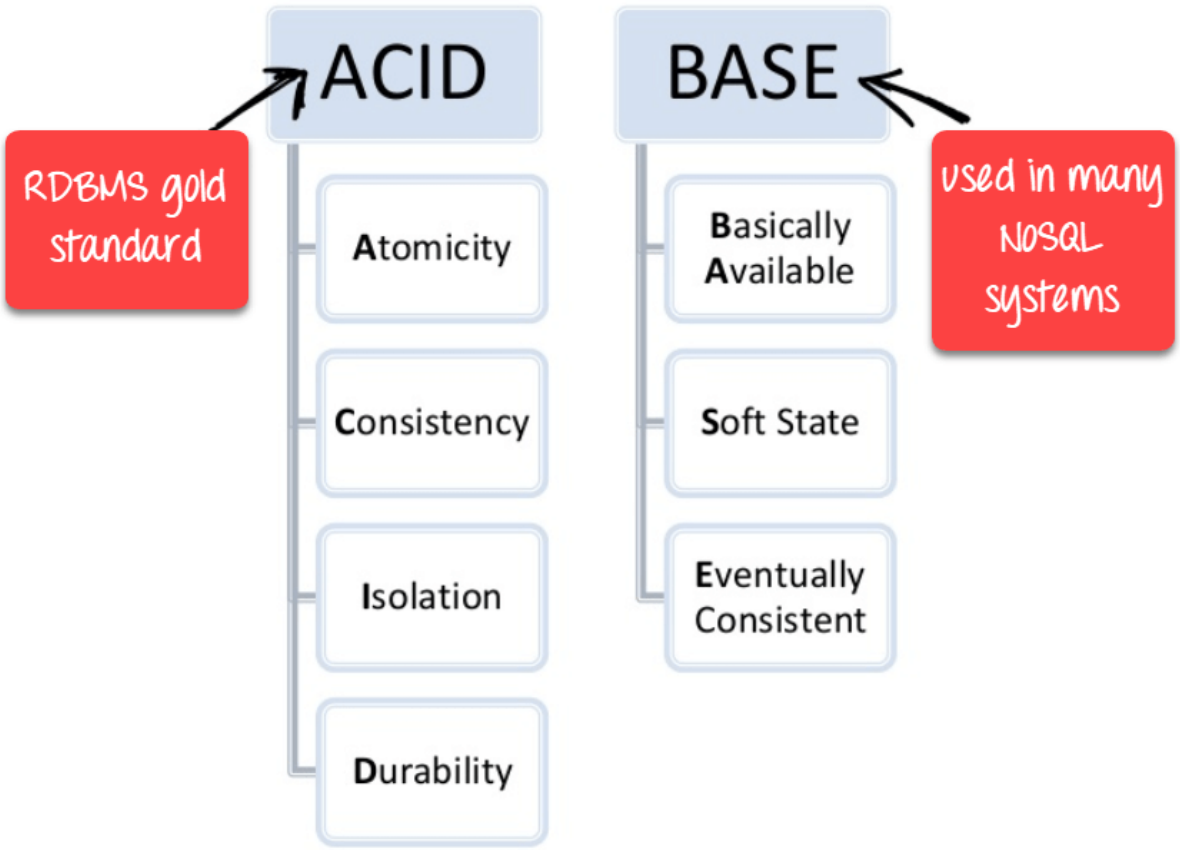
**Consistency:**
The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.

**Availability:**
The database should always be available and responsive. It should not have any downtime.

**Partition Tolerance:**
Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

BASE: Basically Available, Soft state, Eventual consistency

- Basically, available means DB is available all the time as per CAP theorem
- Soft state means even without an input; the system state may change
- Eventual consistency means that the system will become consistent over time




**Eventual Consistency**
The term “eventual consistency” means to have copies of data on multiple machines to get high availability and scalability. Thus, changes made to any data item on one machine has to be propagated to other replicas.
Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time. These copies may be mutually, but in due course of time, they become consistent. Hence, the name eventual consistency.

**Advantages of NoSQL**

- Can be used as Primary or Analytic Data Source
- Big Data Capability
- No Single Point of Failure
- Easy Replication
- No Need for Separate Caching Layer
- It provides fast performance and horizontal scalability.

- Can handle structured, semi-structured, and unstructured data with equal effect
- Object-oriented programming which is easy to use and flexible
- NoSQL databases don't need a dedicated high-performance server
- Support Key Developer Languages and Platforms
- Simple to implement than using RDBMS
- It can serve as the primary data source for online applications.
- Handles big data which manages data velocity, variety, volume, and complexity
- Excels at distributed database and multi-data center operations
- Eliminates the need for a specific caching layer to store data
- Offers a flexible schema design which can easily be altered without downtime or service disruption

**Disadvantages of NoSQL**

- No standardization rules
- Limited query capabilities
- RDBMS databases and tools are comparatively mature
- It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
- When the volume of data increases it is difficult to maintain unique values as keys become difficult
- Doesn't work as well with relational data
- The learning curve is stiff for new developers
- Open source options so not so popular for enterprises.

**Summary**

NoSQL is a non-relational DMS, that does not require a fixed schema, avoids joins, and is easy to scale
The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data
In the year 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
NoSQL databases never follow the relational model it is either schema-free or has relaxed schemas
Four types of NoSQL Database are 1). Key-value Pair Based 2). Column-oriented Graph 3). Graphs based 4). Document-oriented
NOSQL can handle structured, semi-structured, and unstructured data with equal effect
CAP theorem consists of three words Consistency, Availability, and Partition Tolerance
BASE stands for Basically Available, Soft state, Eventual consistency
The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability
NOSQL offer limited query capabilities

## ❖ Value of Relational Database

Relational databases have become such an embedded part of our computing culture that it's easy to take them for granted. It's therefore useful to revisit the benefits they provide.

**Getting at Persistent Data**

Probably the most obvious value of a database is keeping large amounts of persistent data. Most computer architectures have the notion of two areas of memory: a fast volatile “main memory” and a larger but slower “backing store.” Main memory is both limited in space and loses all data when you lose power or something bad happens to the operating system. Therefore, to keep data around, we write it to a backing store, commonly seen as a disk (although these days that disk can be persistent memory). The backing store can be organized in all sorts of ways. For many productivity applications (such as word processors), it’s a file in the file system of the operating system. For most enterprise applications, however, the backing store is a database. The database allows more flexibility than a file system in storing large amounts of data in a way that allows an application program to get at small bits of that information quickly and easily.

**Concurrency**

Enterprise applications tend to have many people looking at the same body of data at once, possibly modifying that data. Most of the time they are working on different areas of that data, but occasionally they operate on the same bit of data. As a result, we have to worry about coordinating these interactions to avoid such things as double booking of hotel rooms. Concurrency is notoriously difficult to get right, with all sorts of errors that can trap even the most careful programmers. Since enterprise applications can have lots of users and other systems all working concurrently, there’s a lot of room for bad things to happen. Relational databases help handle this by controlling all access to their data through transactions. While this isn’t a cure-all (you still have to handle a transactional error when you try to book a room that’s just gone), the transactional mechanism has worked well to contain the complexity of concurrency. Transactions also play a role in error handling. With transactions, you can make a change, and if an error occurs during the processing of the change you can roll back the transaction to clean things up.

**Integration**

Enterprise applications live in a rich ecosystem that requires multiple applications, written by different teams, to collaborate in order to get things done. This kind of inter-application collaboration is awkward because it means pushing the human organizational boundaries. Applications often need to use the same data and updates made through one application have to be visible to others. A common way to do this is shared database integration where multiple applications store their data in a single database. Using a single database allows all the applications to use “each others’ data easily, while the database’s concurrency control handles multiple applications in the same way as it handles multiple users in a single application.

**(Mostly) Standard Model**

Relational databases have succeeded because they provide the core benefits we outlined earlier in a (mostly) standard way. As a result, developers and database professionals can learn the basic relational model and apply it in many projects. Although there are differences between different relational databases, the core mechanisms remain the same: Different vendors’ SQL dialects are similar, transactions operate in mostly the same way.

## ❖ Impedance Mismatch

Impedance mismatch is the term used to refer to the problems that occur due to **differences between the database model and the programming language model**. The practical relational model has 3 components these are:

- Attributes and their data types
- Tuples
- Tables

Problems:
Following problems may occur due to the impedance mismatch:

The first problem that may occur is that **data type mismatch** means the programming language attribute data type may differ from the attribute data type in the data model. Hence it is quite necessary to have a binding for each host programming language that specifies for each attribute type the compatible programming language types. It is necessary to have different data types, for example, we have different data types available in different programming languages such as data types in C are different from Java and both differ from SQL data types.

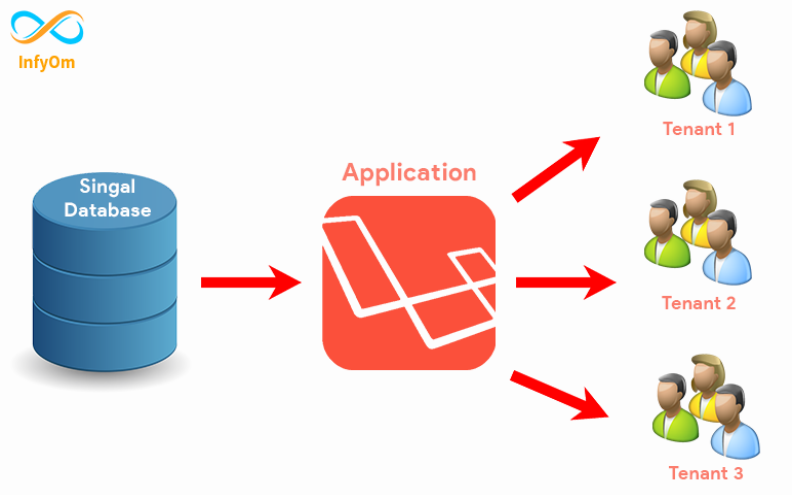
The second problem that may occur is because the **results of most queries are sets or multisets of tuples and each tuple is formed of a sequence of attribute values**. In the program, it is necessary to access the individual data values within individual tuples for printing or processing. Hence there is a need for binding to map the query result data structure which is a table to an appropriate data structure in the programming language. A mechanism is needed to loop over the tuples in a query result in order to access a single tuple at a time and to extract individual values from the tuple. The extracted values are typically copied to appropriate program variables for further processing by the program.

A cursor or iterator is a variable which is used for looping over the tuples in a query result. Individual values within each tuple are extracted into different or unique program variables of the appropriate data type. Impedance mismatch is less of a problem when a special database programming language is designed that uses the same data model and data type as a database model for example Oracles'sPL/SQL.

## ❖ Application and Integration Databases.

### Application Database

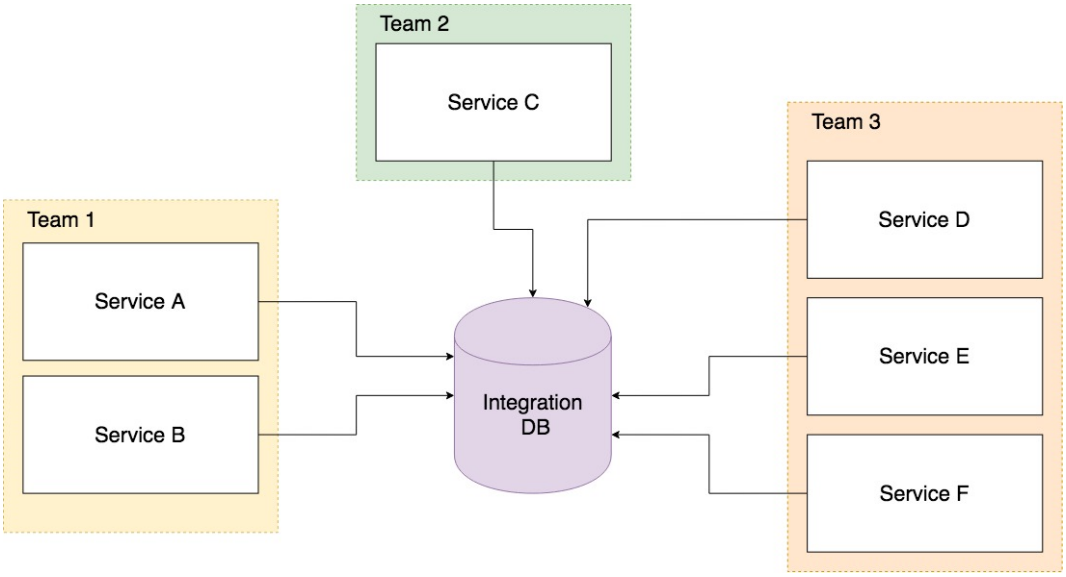
A database that is controlled and accessed by a single application, (in contrast to an IntegrationDatabase). Since only a single application accesses the database, the database can be defined specifically to make that one application's needs easy to satisfy. This leads to a more concrete schema that is usually easier to understand and often less complex than that for an IntegrationDatabase.




### Integration Database

A database serving as a store for numerous applications is called an integration database and therefore, data is integrated across applications. A schema is needed by an integration database, and all applications of clients are taken by the schema into account. Either the resultant schema is general or complicated or both.

Here is an example for a better understanding of the integration database. For example, the computation data of an organization is stored in the Oracle database and client information is stored in Salesforce. The employees can get the integrated data of the two frameworks in a single spot with the help of database integration processes. Website database integration is used by a few organizations for managing and bringing together information from different site pages. Database integration is only viable with the consolidation of data from on-premise systems, legacy systems, and cloud databases. Different software is used by each company.




### ❖ Attack of the clusters.

At the beginning of the new millennium the technology world was hit by the busting of the 1990s dotcom bubble. While this saw many people questioning the economic future of the Internet, the 2000s did see several large web properties dramatically increase in scale. This increase in scale was happening along many dimensions. Websites started tracking activity and structure in a very detailed way. Large sets of data appeared: links, social networks, activity in logs, mapping data. With this growth in data came a growth in users—as the biggest websites grew to be vast estates regularly serving huge numbers of visitors. Coping with the increase in data and traffic required more computing resources. To handle this kind of increase, you have two choices: up or out. Scaling up implies bigger machines, more processors, disk storage, and memory. But bigger machines get more and more expensive, not to mention that there are real limits as your size increases. The alternative is to use lots of small machines in a cluster. A cluster of small machines can use commodity hardware and ends up being cheaper at these kinds of scales. It can also be more resilient—while individual machine failures are common, the overall cluster can be built to keep going despite such failures, providing high reliability.

As large properties moved towards clusters, that revealed a new problem—relational databases are not designed to be run on clusters. Clustered relational databases, such as the Oracle RAC or Microsoft SQL Server, work on the concept of a shared disk subsystem. They use a cluster-aware file system that writes to a highly available disk subsystem—but this means the cluster still has the disk subsystem as a single point of failure. Relational databases could also be run as separate servers for different sets of data, effectively sharding (“Sharding,” p. 38) the database. While this separates the load, all the sharding has to be controlled by the application which has to keep track of which database server to talk to for each bit of data. Also, we lose any querying, referential integrity,

transactions, or consistency controls that cross shards. A phrase we often hear in this context from people who've done this is "unnatural acts." These technical issues are exacerbated by licensing costs. Commercial relational databases are usually priced on a single-server assumption, so running on a cluster raised prices and led to frustrating negotiations with purchasing departments. This mismatch between relational databases and clusters led some organization to consider an alternative route to data storage. Two companies in particular—Google and Amazon—have been very influential. Both were on the forefront of running large clusters of this kind; furthermore, they were capturing huge amounts of data. These things gave them the motive. Both were successful an growing companies with strong technical components, which gave them the means and opportunity. It was no wonder they had murder in mind for their relational databases. As the 2000s drew on, both companies produced brief but highly influential papers about their efforts: BigTable from Google and Dynamo from Amazon.

It's often said that Amazon and Google operate at scales far removed from most organizations, so the solutions they needed may not be relevant to an average organization. While it's true that most software projects don't need that level of scale, it's also true that more and more organizations ar beginning to explore what they can do by capturing and processing more data—and to run into the same problems. So, as more information leaked out about what Google and Amazon had done, people began to explore making databases along similar lines—explicitly designed to live in a world of clusters. While the earlier menaces to relational dominance turned out to be phantoms, the threat from clusters was serious.

## ❖ Emergence of NoSQL

- ➢ The term "NoSQL" first made its appearance in the late 90s as the name of an open-source relational database [Strozzi NoSQL]. Led by Carlo Strozzi, this database stores its tables as ASCII files, each tuple represented by a line with fields separated by tabs.
- ➢ The usage of "NoSQL" that we recognize today traces back to a meetup on June 11, 2009 in San Francisco organized by Johan Oskarsson, a software developer based in London. The example of BigTable and Dynamo had inspired a bunch of projects experimenting with alternative data storage, and discussions of these had become a feature of the better software conferences around that time. Johan was interested in finding out more about some of these new databases while he was in San Francisco for a Hadoop summit. Since he had little time there, he felt that it wouldn't be feasible to visit them all, so he decided to host a meetup where they could all come together and present their work to whoever was interested. Johan wanted a name for the meetup—something that would make a good Twitter hashtag: short, memorable, and without too many Google hits so that a search on the name would quickly find the meetup. He asked for suggestions on the #cassandra IRC channel and got a few, selecting the suggestion of "NoSQL" from Eric Evans (a developer at Rackspace, no connection to the DDD Eric Evans). While it had the disadvantage of being negative and not really describing these systems, it did fit the hashtag criteria. At the time they were thinking of only naming a single meeting and were not expecting it to catch on to name this entire technology trend [Oskarsson].
- ➢ The term "NoSQL" caught on like wildfire, but it's never been a term that's had much in the way of a strong definition. The original call [NoSQL Meetup] for the meetup asked for "open-source, distributed, non relational databases."
- ➢ To begin with, there is the obvious point that NoSQL databases don't use SQL. Some of them do have query languages, and it makes sense for them to be similar to SQL in order to make them easier to learn. Cassandra's CQL is like this—"exactly like SQL (except where it's not)" [CQL]. But so far none have implemented anything that would fit even the rather flexible notion of standard SQL.
- ➢ Most NoSQL databases are driven by the need to run on clusters, and this is certainly true of those that were talked about during the initial meetup. This has an effect on their data model as well as

their approach to consistency. Relational databases use ACID transactions to handle consistency across the whole database.

## ❖ Key Points

- ❖ Relational databases have been a successful technology for twenty years, providing persistence, concurrency control, and an integration mechanism.
- ❖ Application developers have been frustrated with the impedance mismatch between the relational model and the in-memory data structures.
- ❖ There is a movement away from using databases as integration points towards encapsulating databases within applications and integrating through services.
- ❖ The vital factor for a change in data storage was the need to support large volumes of data by running on clusters. Relational databases are not designed to run efficiently on clusters.
- ❖ NoSQL is an accidental neologism. There is no prescriptive definition—all you can make is an observation of common characteristics.
- ❖ The common characteristics of NoSQL databases are
- ❖ Not using the relational model
- ❖ Running well on clusters
- ❖ Open-source
- ❖ Built for the 21st century web estates
- ❖ Schemaless
- ❖ The most important result of the rise of NoSQL is Polyglot Persistence.