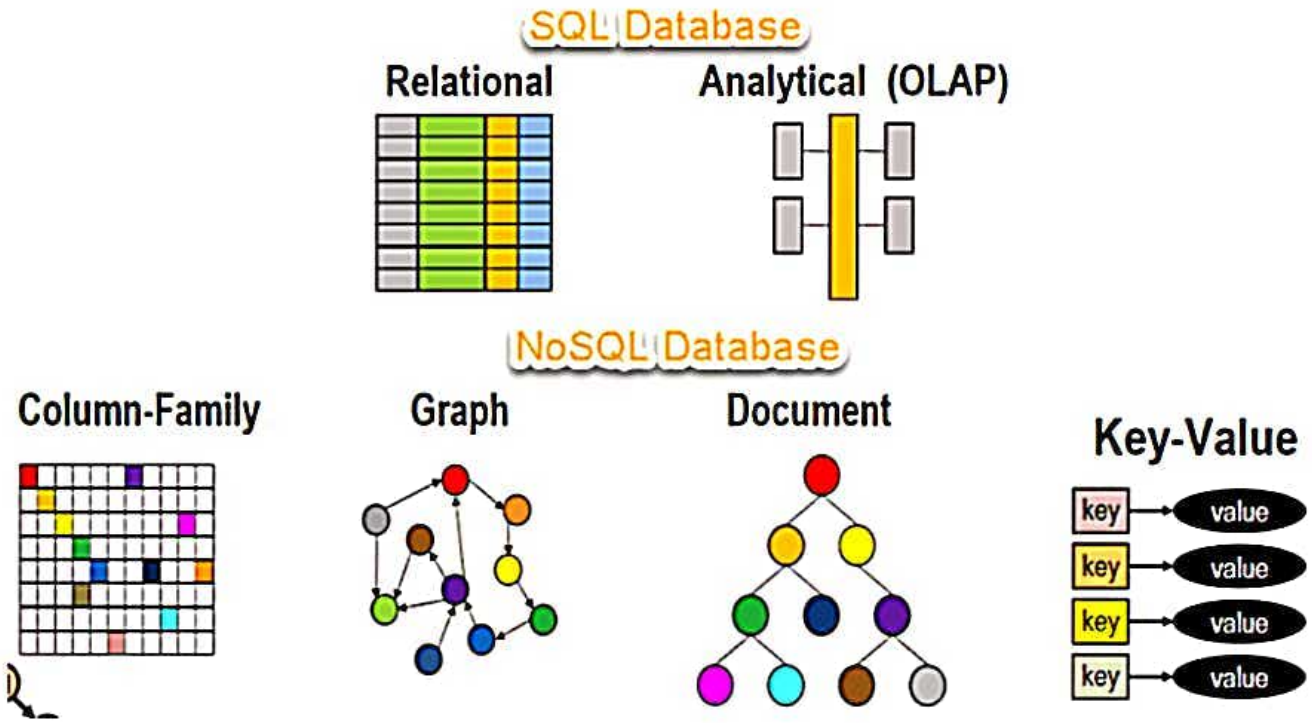
1 ❖ **Overview of NoSQL**

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:



Differeneces between NoSQL and Relational database

NoSQL Database	Relational Database
NoSQL Database supports a very simple query language.	Relational Database supports a powerful query language.
NoSQL Database has no fixed schema.	Relational Database has a fixed schema.
NoSQL Database is only eventually consistent.	Relational Database follows acid properties. (Atomicity, Consistency, Isolation, and Durability)
NoSQL databases don't support transactions (support only simple transactions).	Relational Database supports transactions (also complex transactions with joins).
NoSQL Database is used to handle data coming in high velocity.	Relational Database is used to handle data coming in low velocity.
The NoSQL?s data arrive from many locations.	Data in relational database arrive from one or few locations.
NoSQL database can manage structured, unstructured and semi-structured data.	Relational database manages only structured data.
NoSQL databases have no single point of failure.	Relational databases have a single point of failure with failover.
NoSQL databases can handle big data or data in a very high volume .	NoSQL databases are used to handle moderate volume of data.
NoSQL has decentralized structure.	Relational database has centralized structure.
NoSQL database gives both read and write scalability.	Relational database gives read scalability only.
NoSQL database is deployed in horizontal fashion.	Relation database is deployed in vertical fashion.

◆ Emergence of NoSQL

1b

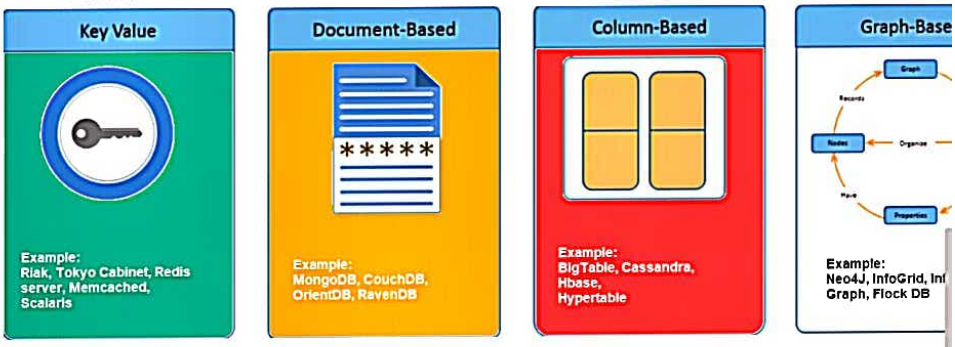
- The term "NoSQL" first made its appearance in the late 90s as the name of an open-source relational database [Strozzi NoSQL]. Led by Carlo Strozzi, this database stores its tables as ASCII files, each tuple represented by a line with fields separated by tabs.
- The usage of "NoSQL" that we recognize 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.

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.

ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

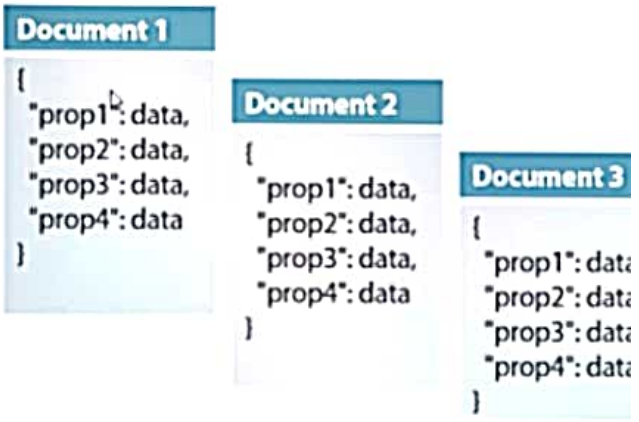
Column based NoSQL database

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.

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data



Relational Vs. Document

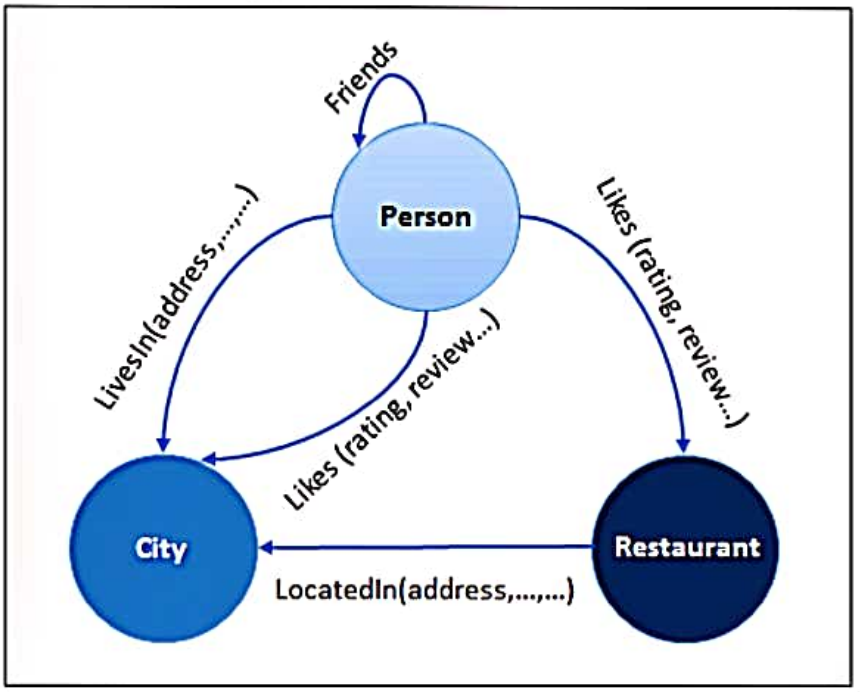
In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. 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.

3a

Relational Database Management Systems (RDBMS) and NoSQL databases serve different purposes and have distinct characteristics. While RDBMS has been a dominant technology for many years, NoSQL databases have gained popularity due to their flexibility and scalability. Here are some limitations of RDBMS compared to NoSQL databases:

1. **Schema Flexibility:**
   - RDBMS: Requires a predefined schema with a fixed structure. Any changes to the schema can be complex and may require downtime.
   - NoSQL: Provides schema flexibility, allowing for dynamic and evolving data structures. This is particularly beneficial for applications with changing data requirements.
2. **Scalability:**
   - RDBMS: Traditional RDBMS systems may face challenges in horizontal scaling. Scaling vertically (adding more powerful hardware) has limitations.
   - NoSQL: Designed to scale horizontally, allowing for easy distribution across multiple servers or nodes. This makes it well-suited for handling large amounts of data and high traffic loads.
3. **Data Model:**
   - RDBMS: Primarily supports the relational data model (tables with rows and columns), which may not be the best fit for certain types of data, such as hierarchical or nested data.
   - NoSQL: Offers a variety of data models, including document-based (like MongoDB), key-value pairs (like Redis), column-family, and graph databases. This flexibility allows developers to choose a model that best suits their data.
4. **Complex Transactions:**
   - RDBMS: Well-suited for complex transactions with ACID properties (Atomicity, Consistency, Isolation, Durability). However, this can impact performance in distributed and highly scalable environments.
   - NoSQL: Emphasizes eventual consistency over strong consistency. This is suitable for applications where some level of inconsistency is acceptable, and scalability is a higher priority.

5. **Performance for Certain Workloads:**
   - RDBMS: Optimized for complex query processing and joins, which can result in slower performance for simple read and write operations.
   - NoSQL: Often designed for high-throughput and low-latency access to data, making them more efficient for certain types of workloads, such as read-heavy or write-heavy operations.
6. **Cost and Licensing:**
   - RDBMS: Commercial RDBMS systems may involve significant licensing costs, especially as the scale of the infrastructure grows.
   - NoSQL: Many NoSQL databases are open-source, reducing licensing costs. This can be advantageous for startups and organizations with budget constraints.
7. **Consistency Trade-offs:**
   - RDBMS: Maintains strong consistency but may face challenges in distributed environments, impacting performance.
   - NoSQL: Emphasizes eventual consistency, allowing for greater scalability but at the cost of immediate consistency in some scenarios.

It's important to note that the choice between RDBMS and NoSQL depends on the specific requirements and characteristics of the application being developed. Each type of database system has its strengths and weaknesses, and the decision should be based on the specific needs of the project.

◆ Attack of the clusters.

3b

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,

---

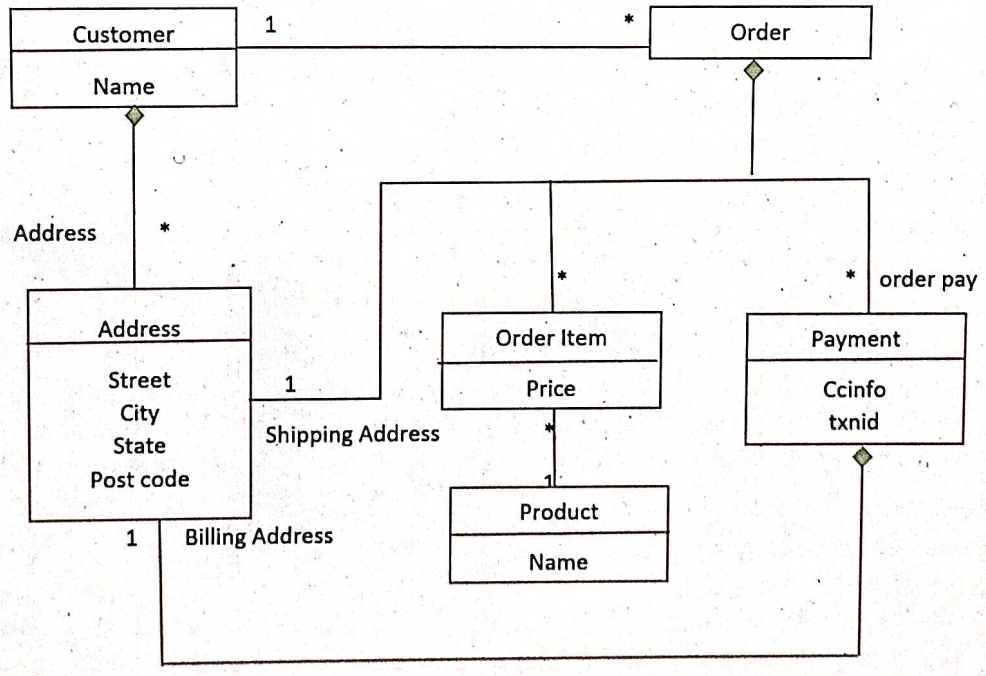
transactions, or consistency controls that cross shards. A phrase we often hear in this context from people who’ve done this is “unnatural acts.” These technical issues are exacerbated by licensing costs. Commercial relational databases are usually priced on a single-server assumption, so running on 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.

Aggregate Data Models: 4 , 7b

The term aggregate means a collection of objects that we use to treat as a unit. An aggregate is a collection of data that we interact with as a unit. These units of data or aggregates form the boundaries for ACID operation.

Example of Aggregate Data Model:



Here in the diagram have two Aggregate:

- Customer and Orders link between them represent an aggregate.
- The diamond shows how data fit into the aggregate structure.
- Customer contains a list of billing address
- Payment also contains the billing address
- The address appears three times and it is copied each time
- The domain is fit where we don't want to change shipping and billing address.

Consequences of Aggregate Orientation:

- Aggregation is not a logical data property It is all about how the data is being used by applications.
- An aggregate structure may be an obstacle for others but help with some data interactions.
- It has an important consequence for transactions.
- NoSQL databases don't support ACID transactions thus sacrificing consistency.
- aggregate-oriented databases support the atomic manipulation of a single aggregate at a time.

Advantage:

- It can be used as a primary data source for online applications.
- Easy Replication.
- No single point Failure.
- It provides fast performance and horizontal Scalability.
- It can handle Structured semi-structured and unstructured data with equal effort.

Disadvantage:

- No standard rules.
- Limited query capabilities.
- Doesn't work well with relational data.
- Not so popular in the enterprise.
- When the value of data increases it is difficult to maintain unique values.

Relations:

In the context of databases, a relation refers to the logical association or connection between two or more entities. The term is most commonly associated with relational database management systems (RDBMS). In a relational database, data is organized into tables, and the relationships between these tables are established using keys.

- **Tables:** In an RDBMS, a relation is often represented as a table. Each table consists of rows and columns, where each row represents a record, and each column represents a specific attribute or field.
- **Keys:** Relations between tables are established using keys. A key is a column or set of columns that uniquely identifies each record in a table. The primary key is a special type of key that uniquely identifies each record in the table.
- **Foreign Keys:** A foreign key is a column in one table that refers to the primary key in another table. It establishes a link between the two tables, allowing data in one table to be related to data in another.

Example:
Consider two tables - "Employees" and "Departments." The "Employees" table might have a foreign key column called "DepartmentID" that references the "DepartmentID" column in the "Departments" table. This establishes a relationship between the employees and their respective departments.

Aggregates:

In the context of databases and queries, aggregates refer to functions that perform a calculation on a set of values and return a single value as a result. Aggregates are commonly used in conjunction with GROUP BY clauses to summarize and analyze data.

- **Common Aggregate Functions:**
  - **SUM:** Calculates the sum of values in a column.
  - **AVG:** Computes the average of values in a column.
  - **COUNT:** Counts the number of rows in a group.
  - **MAX:** Finds the maximum value in a column.
  - **MIN:** Finds the minimum value in a column.
- **GROUP BY Clause:** Aggregates are often used with the GROUP BY clause to group rows based on certain criteria. This allows the aggregate functions to operate on each group separately.

ACID Model: 8a

Collections of operations that form a single logical unit of work are called transactions and the database system must ensure proper execution of transactions and the ACID database transaction model ensures that a performed transaction is always consistent. To explain ACID in more detail and easy way is to understand through breaking down the acronym, ACID:

1. **Atomicity:** This property states transaction must be treated as an atomic unit, that is, either all of its operations are executed or none, and there must be no state in a database where a transaction is left partially completed also the states should be defined either before the execution of the transaction or after the execution of the transaction.
2. **Consistency:** The database must remain in a consistent state after any transaction also no transaction should have any adverse effect on the data residing in the database and if the database was in a consistent state before the execution of a transaction then it must remain consistent after the execution of the transaction as well.
3. **Isolation:** In a database system where more than one transaction is being executed simultaneously and in parallel, the property of isolation states that each one of the transactions is going to be administered and executed as it is the only transaction in the system also no transaction will affect the existence of any other transactions.
4. **Durability:** The database should be durable enough to hold all its latest updates even if the system fails or restarts so, In a practical way of saying that if a transaction updates a chunk of data in a database and commit is performed then the database will hold the modified data but if the commit is not performed then no data is modified and it can only be done when the system start.

Base Model:

The rise in popularity of NoSQL databases provided a flexible and fluidity with ease to manipulate data and as a result, a new database model was designed, reflecting these properties. The acronym BASE is slightly more confusing than ACID but however, the words behind it suggest ways in which the BASE model is different and acronym BASE stands for:-

1. **Basically Available:** Instead of making it compulsory for immediate consistency, BASE-modelled NoSQL databases will ensure the availability of data by spreading and replicating it across the nodes of the database cluster.
2. **Soft State:** Due to the lack of immediate consistency, the data values may change over time. The BASE model breaks off with the concept of a database that obligates its own consistency, delegating that responsibility to developers.
3. **Eventually Consistent:** The fact that BASE does not obligates immediate consistency but it does not mean that it never achieves it. However, until it does, the data reads are still possible (even though they might not reflect reality).

Difference between ACID and BASE:

S. No	Criteria	ACID	BASE
1.	Simplicity	Simple	Complex
2.	Focus	Commits	Best attempt
3.	Maintenance	High	Low
4.	Consistency Of Data	Strong	Weak/Loose
5.	Concurrency scheme	Nested Transactions	Close to answer
6.	Scaling	Vertical	Horizontal
7.	Implementation	Easy to implement	Difficult to implement
8.	Upgrade	Harder to upgrade	Easy to upgrade
9.	Type of database	Robust	Simple
10.	Type of code	Simple	Harder
11.	Time required for completion	Less time	More time.
12.	Examples	Oracle, MySQL, SQL Server, etc.	DynamoDB, Cassandra, CouchDB, SimpleDB etc.

8b **Characteristics of NoSQL Databases:**

1. **Schema-less or Schema-flexible:**
   - NoSQL databases often allow for a flexible or schema-less approach, where data can be inserted without a predefined schema. This flexibility is beneficial for handling diverse and evolving data structures.
2. **Horizontal Scalability:**
   - NoSQL databases are designed to scale horizontally, allowing the addition of more servers or nodes to handle increased data and traffic. This makes them suitable for large-scale distributed systems.
3. **Diverse Data Models:**
   - NoSQL databases support various data models, including document-based (e.g., MongoDB), key-value pairs (e.g., Redis), column-family (e.g., Apache Cassandra), and graph databases (e.g., Neo4j). This diversity allows developers to choose a data model that fits their application requirements.
4. **High Performance for Read and Write Operations:**
   - NoSQL databases are optimized for specific use cases, such as high-throughput read and write operations. This can make them well-suited for scenarios where low-latency access to data is crucial.
5. **No Complex Joins:**
   - Unlike traditional relational databases, NoSQL databases often avoid complex joins. Instead, they may denormalize data to improve read performance. This simplifies queries and enhances performance in certain scenarios.
6. **BASE (Basically Available, Soft state, Eventually consistent):**
   - NoSQL databases often follow the BASE model, which provides more relaxed consistency compared to the strict ACID properties of traditional RDBMS. This trade-off allows for improved availability and fault tolerance.

Applications of NoSQL Databases:

1. **Big Data and Real-time Applications:**
   - NoSQL databases are well-suited for handling large volumes of data generated in real-time, making them suitable for big data analytics and applications requiring low-latency processing.
2. **Content Management Systems (CMS):**
   - NoSQL databases, especially document-based ones like MongoDB, are used in content management systems where flexibility in document structures and quick retrieval are essential.
3. **Caching and Session Storage:**
   - Key-value stores like Redis are often employed for caching frequently accessed data or storing session information in web applications, providing fast and efficient data retrieval.
4. **IoT (Internet of Things) Applications:**
   - NoSQL databases are used in IoT scenarios where data from a large number of devices needs to be stored, processed, and analyzed. The flexibility of NoSQL databases accommodates the varied data generated by IoT devices.
5. **Social Media and User Profiles:**
   - Graph databases like Neo4j are employed for representing and querying relationships in social networks. NoSQL databases can efficiently store and retrieve user profile data in social media applications.
6. **E-commerce and Recommendation Engines:**
   - NoSQL databases can handle the diverse and dynamic nature of product catalogs in e-commerce applications. They are also used in recommendation engines to store and process user behavior data.

In summary, NoSQL databases offer flexibility, scalability, and performance advantages, making them suitable for a variety of applications, especially those involving large-scale data, real-time processing, and diverse data structures.