# جامعة طرابلس

# كلية تقنية المعلومات

## قواعد البيانات النقالة والغير متجانسة

## Heterogeneous and Mobile Databases

## **ITMC322**

**أستاذ المادة / محمد أوهيبة**

**المحاضرة الرابعة**

# Review of topics

***•Getting into the NoSQL movement***

***-Introduction***

***-Comparing RDBMS and NoSQL databases***

***-Transactions in both RDBMS and NON RDBMS***

***-Managing read-write concurrency***

***- MongoDB core elements***

# Getting into the NoSQL movement

**Introduction**

- NoSQL is a generic term used to refer to any data store that does not follow the traditional RDBMS model.

- the data is nonrelational and it generally does not use SQL as a query language.
- 
- Most of the databases that are categorized as NoSQL focus on availability and scalability in spite of atomicity or consistency

# Getting into the NoSQL movement (Cont.)

all databases that fall into this category have some characteristics in common such as:

•**Storing data in many formats**: Almost all RDBMS databases are based on the storage or rows in tables. NoSQL databases, on the other hand, can use different formats such as document stores, graph databases, key-value stores and even more.

**Joinless**: NoSQL databases are able to extract your data using simple document-oriented interfaces without using SQL joins.

**Schemaless data representation**: A characteristic of NoSQL implementations is that they are based on a schemaless data representation. you don't need to define a data structure beforehand, which can thus continue to change over time.

# Getting into the NoSQL movement (Cont.)

•**Ability to work with many machines**: Most NoSQL systems buy you the ability to store your database on multiple machines while maintaining high-speed performance.

database transactions should be:

- **Atomicity**: Everything in a transaction either succeeds or is rolled back
- **Consistency**: Every transaction must leave the database in a consistent state
- **Isolation**: Each transaction that is running cannot interfere with other transactions
- **Durability**: A completed transaction gets persisted, even after applications restart

# Getting into the NoSQL movement (Cont.)

Essential requirements when designing applications for distributed architectures :

**Consistency**: This means the database mostly remains adherent to its rules (constraints, triggers, and so on) after the execution of each operation and that any future transaction will see the effects of the earlier transactions committed. For example, after executing an update, all the clients see the same data.

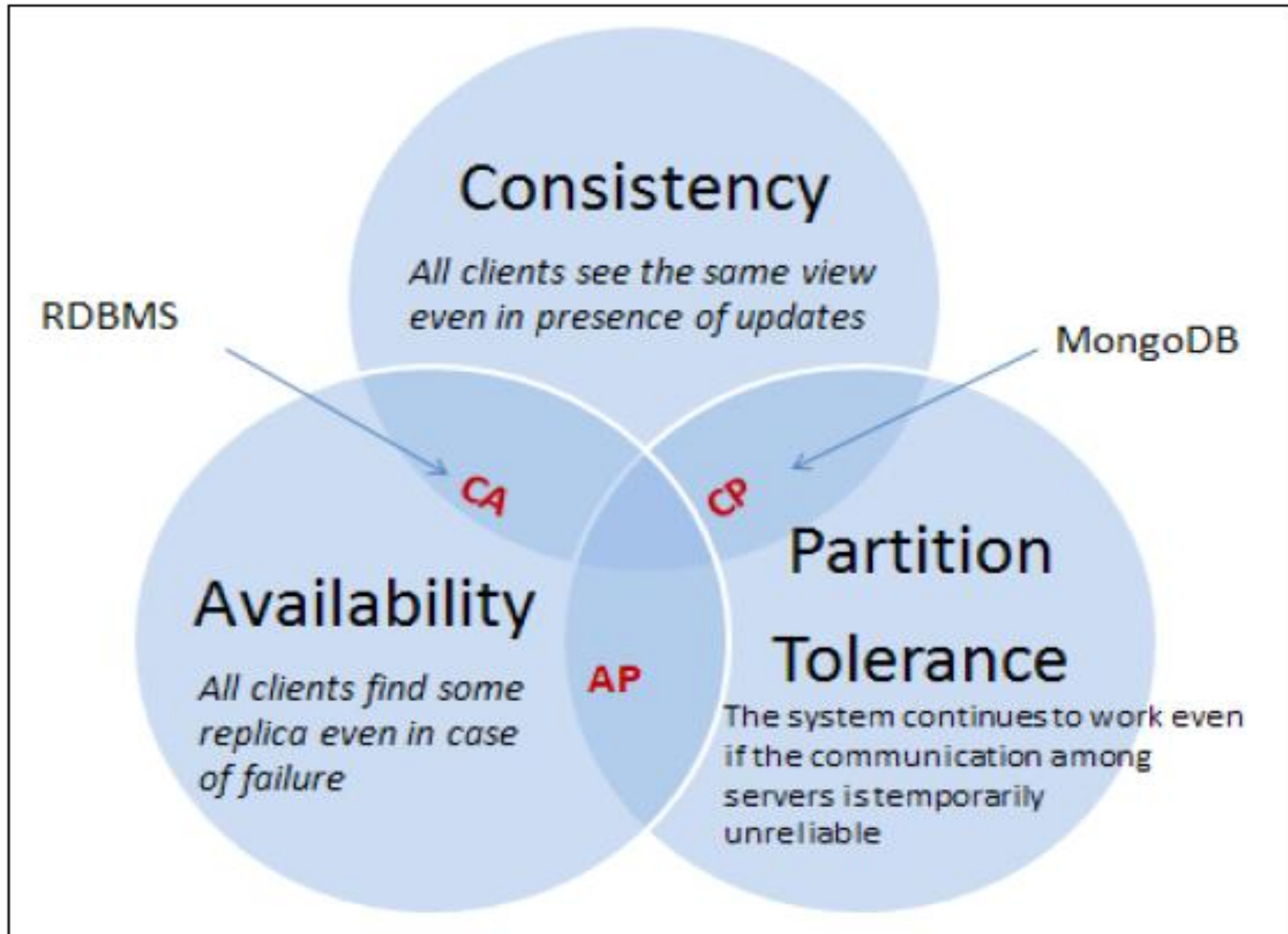
**Availability**: Each operation is guaranteed a response—a successful or failed execution. This, in practice, means no downtime.

# Getting into the NoSQL movement (Cont.)

**Partition tolerance**: This means the system continues to function even if the communication among the servers is temporarily unreliable (for example, the servers involved in the transaction may be partitioned into multiple groups, which cannot communicate with one another).

In practice, as it is theoretically impossible to have all three requirements met, a combination of two must be chosen and this is usually the deciding factor in what technology is used, as shown in the following figure:

# Getting into the NoSQL movement (Cont.)



# Getting into the NoSQL movement (Cont.)

•If you are designing a typical web application that uses a SQL database, most likely, you are in the CA part of the diagram. This is because a traditional RDBMS is typically transaction-based (C) and it can be highly available (A). However, it cannot be Partition Tolerance (P) because SQL databases tend to run on single nodes.

•MongoDB, on the other hand, is consistent by default (C). This means if you perform a write on the database followed by a read, you will be able to read the same data (assuming that the write was successful).
Besides consistency, MongoDB leverages Partition Tolerance (P) by means of replica sets. In a replica set, there exists a single primary node that accepts writes, and asynchronously replicates a log of its operations to other secondary databases.

# Getting into the NoSQL movement (Cont.)

- However, not all NoSQL databases are built with the same focus. An example of this is CouchDB. Just like MongoDB, it is document oriented and has been built to scale across multiple nodes easily; on the other hand, while MongoDB (CP) favors consistency, CouchDB favors availability (AP) in spite of consistency.

- The following table summarizes the most common NoSQL databases and their position relative to CAP attributes:

| **Database** | **Consistent, Partition-Tolerant (CP)** | **Available, Partition-Tolerant (AP)** |
| --- | --- | --- |
| BigTable | X | |
| Hypertable | X | |
| HBase | X | |
| MongoDB | X | |
| Terrastore | X | |
| Redis | X | |
| Scalaris | X | |
| MemcacheDB | X | |
| Berkeley DB | X | |
| Dynamo | | X |
| Voldemort | | X |
| Tokyo Cabinet | | X |
| KAI | | X |
| Cassandra | | X |
| CouchDB | | X |
| SimpleDB | | X |
| Riak | | X |

# Comparing RDBMS and NoSQL databases

•we can identify a set of pros and cons related to each technology. This can lead to a better understanding of which one is most fit for our scenarios. Let's start from traditional RDBMS:.

| **RDBMS pros** | **RDBMS cons** |
|---|---|
| ACID transactions at the database level make development easier. | The object-relational mapping layer can be complex. |
| Fine-grained security on columns and rows using views prevents views and changes by unauthorized users. Most SQL code is portable to other SQL databases, including open source options. | RDBMS doesn't scale out when joins are required. |
| Typed columns and constraints will validate data before it's added to the database and increase data quality. | Sharding over many servers can be done but requires application code and will be operationally inefficient. |
| The existing staff members are already familiar with entity-relational design and SQL. | Full-text search requires third-party tools. |
| Well-consolidated theoretical basis and design rules. | Storing high-variability data in tables can be challenging. |

# Comparing RDBMS and NoSQL databases

The following is a table that contains the advantages and disadvantages of NoSQL databases:

| NoSQL pros | NoSQL cons |
|---|---|
| It can store complex data types (such as documents) in a single item of storage. | There is a lack of server-side transactions; therefore, it is not fit for inherently transactional systems. |
| It allows horizontal scalability, which does not require you to set up complex joins and data can be easily partitioned and processed in parallel. | Document stores do not provide fine-grained security at the element level. |
| It saves on development time as it is not required to design a fine-grained data model. | NoSQL systems are new to many staff members and additional training may be required. |
| It is quite fast for inserting new data and for simple operations or queries. | The document store has its own proprietary nonstandard query language, which prohibits portability. |
| It provides support for Map/Reduce, which is a simple paradigm that allows for scaling computation on a cluster of computing nodes. | There is an absence of standardization. No standard APIs or query languages. It means that migration to a solution from different vendors is more costly. Also, there are no standard tools (for example, for reporting). |

# Transactions in both RDBMS and NON RDBMS

- With an RDBMS, you can update the database in sophisticated ways using SQL and wrap multiple statements in a transaction to get atomicity and rollback. MongoDB doesn't support transactions. This is a solid tradeoff based on MongoDB's goal of being simple, fast, and scalable. MongoDB, however, supports a range of atomic update operations that can work on the internal structures of a complex document. So, for example, by including multiple structures within one document (such as arrays), you can achieve an update in a single atomic way, just like you would do with an ordinary transaction.

# Transactions in both RDBMS and NON RDBMS

- Application's requirements can be met via document updates (also by using nested documents to provide an atomic update), then this is a perfect use case for MongoDB, which will allow a much easier horizontal scaling of your application.
On the other hand, if strict transaction semantics (such as a banking application) are required, then nothing can beat a relational database. In some scenarios, you can combine both approaches (RDBMS and MongoDB) to get the best of both worlds, at the price of a more complex infrastructure to maintain. Such hybrid solutions are quite common; however, you can see them in production apps such as the New York Times website.

# Managing read-write concurrency

•In RDBMS, managing the execution of concurrent work units is a fundamental concept. The underlying implementation of each database uses behind the scenes locks or Multiversion control to provide the isolation of each work unit. On the other hand, MongoDB uses reader/writer locks that allow concurrent readers shared access to a resource, such as a database or collection, but give exclusive access to a single write operation. In more detail here is how MongoDB handles read and write locks:

• There can be an unlimited number of simultaneous readers on a database
• There can only be one writer at a time on any collection in any one database
• The writers block out the readers once a write request comes in; all the readers are blocked until the write completes (which is also known as writer-greedy)

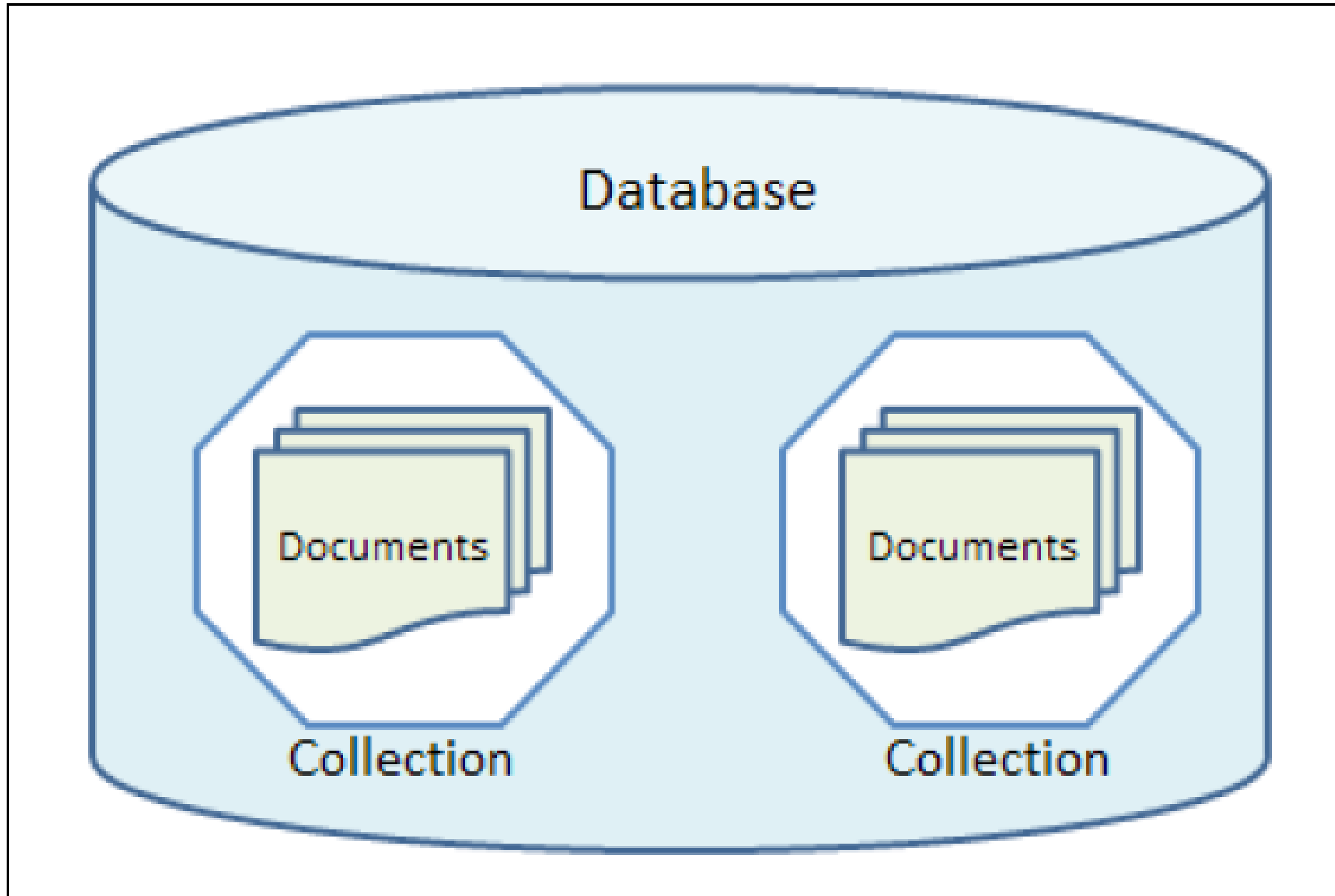
# MongoDB core elements

•The core elements the database is composed of. Actually, MongoDB is organized with a set of building blocks, which include the following:

•**Database**: This is, just like for the database, the top-level element. However, a relational database contains (mostly) tables and views. A Mongo Database, on the other hand, is a physical container of a structure called a collection.
•**Collection**: This is a set of MongoDB documents. A collection is the equivalent of an RDBMS table. There can be only one collection with that name on the database but obviously multiple collections can coexist in a database.
•**Documents**: This is the most basic unit of data in MongoDB. Basically, it is composed by a set of key-value pairs. Unlike database records, documents have a dynamic schema, which means documents that are part of the same collection do not need to have the same set of fields.

# MongoDB core elements

The following diagram summarizes the concepts



# The End

# Thanks for listening ..

# Any questions ?